(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 February 2003 (20.02.2003)

PCT

(10) International Publication Number
WO 03/014921 A1

(51) International Patent Classification⁷: G06F 9/44

(21) International Application Number: PCT/US02/20992

(22) International Filing Date: 2 July 2002 (02.07.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/302,891	2 July 2001 (02.07.2001)	US
60/306,376	17 July 2001 (17.07.2001)	US
10/187,858	27 June 2002 (27.06.2002)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/302,891 (CIP)
Filed on 2 July 2001 (02.07.2001)

(71) Applicants *(for all designated States except US)*: NAZOMI COMMUNICATIONS, INC. [US/US]; 2200 Laurelwood Road, Santa Clara, CA 95054 (US). PATEL, Mukesh, K. [US/US]; 787 Boar Circle, Fremont, CA 94539 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: HILLMAN, Dan [US/US]; 6607 Winterset Way, San Jose, CA 95120 (US). KAMDAR, Jay [US/US]; 10080 Carmen Road, Cupertino, CA 95014 (US). SHIELL, Jon [US/US]; 801 Seabury Drive, San Jose, CA 95136 (US). RAVAL, Udaykumar, R. [IN/US]; 2200 Monroe Street, #1608, Santa Clara, CA 95050 (US).

(74) Agent: O'MALLEY, Joseph, P.; Burns, Doane, Swecker & Mathis, LLP, P.O BOX 1404, Alexandria, VA 22313 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 03/014921 A1

(54) Title: INTERMEDIATE LANGUAGE ACCELERATOR CHIP

(57) Abstract: An accelerator chip can be positioned between a processor chip and a memory (26). The accelerator chip enhances the operation of a Java program by running portions (22) of the Java program for the processor chip (24). In a preferred embodiment, the accelerator chip includes a hardware translator unit and a dedicated execution engine.

INTERMEDIATE LANGUAGE ACCELERATOR CHIP

Background of the Invention

[001] Java™ is an object-orientated programming language developed by Sun Microsystems. The Java language is small, simple and portable across platforms and operating systems, both at the source and binary level. This makes the Java programming language very popular on the Internet.

[002] Java's platform independence and code compaction are the most significant advantages of Java over conventional programming languages. In conventional programming languages, the source code of a program is sent to a compiler which translates the program into machine code or processor instructions. The processor instructions are native to the system's processor. If the code is compiled on an Intel-based system, the resulting program will run only on other Intel-based systems. If it is desired to run the program on another system, the user must go back to the original source code, obtain a compiler for the new processor, and recompile the program into the machine code specific to that other processor.

[003] Java operates differently. The Java compiler takes a Java program and, instead of generating machine code for a specific processor, generates bytecodes. Bytecodes are instructions that look like machine code, but are not specific to any processor. To execute a Java program, a bytecode interpreter takes the Java bytecodes and converts them to equivalent native processor instructions and executes the Java program. The Java bytecode interpreter is one component of the Java Virtual Machine (JVM).

[004] Having the Java programs in bytecode form means that instead of being specific to any one system, the programs can be run on any platform and any operating system as long as a Java Virtual Machine is available. This allows a binary bytecode file to be executable across platforms.

[005] The disadvantage of using bytecodes is execution speed. System- specific programs that run directly on the hardware from which they are compiled run significantly faster than Java bytecodes, which must be processed by the Java Virtual Machine. The processor must both convert the Java bytecodes into native instructions in the Java Virtual Machine and execute the native instructions.

[006] Poor Java software performance, particularly in embedded system designs, is a well-known issue and several techniques have been introduced to increase performance. However these techniques introduce other undesirable side effects. The most common techniques include increasing system and/or microprocessor clock frequency, modifying a JVM to compile Java bytecodes and using a dedicated Java microprocessor.

[007] Increasing a microprocessor's clock frequency results in overall improved system performance gains, including performance gains in executing Java software. However, frequency increases do not result in one-for-one increases in Java software performance. Frequency increases also raise power consumption and overall system costs. In other words, clocking a microprocessor at a higher frequency is an inefficient method of accelerating Java software performance.

[008] Compilation techniques (e.g., just in time "JIT" compilation) contribute to erratic performance because the speed of software execution is delayed during compilation. Compilation also increases system memory usage because compiling and storing a Java program consumes an additional five to ten times the amount of memory over what is required to store the original Java program.

[009] Dedicated Java microprocessors use Java bytecode instructions as their native language, and while they execute Java software with better performance than typical commercial microprocessors they impose several significant design constraints. Using a dedicated Java microprocessor requires the system design to revolve around it and forces the utilization of specific development tools usually only available from the Java microprocessor vendor. Furthermore, all operating system software and device drivers must be custom developed from scratch because commercial software of this nature does not exist.

[0010] It is desired to have an embedded system with improved Java software performance.

Summary of the Present Invention

[0011] One embodiment of the present invention comprises a system including at least one memory, a processor chip operably connected to the one memory, and an Accelerator Chip. The memory access for the processor chip to at least one memory being sent through the

Accelerator Chip. The Accelerator Chip has direct access to the at least one memory. The Accelerator Chip is adapted to run at least portions of programs using intermediate language instructions. The intermediate language instructions include Java bytecodes and also include the intermediate language forms of other interpreted languages. These intermediate language forms include Multos bytecodes, UCSD Pascal P-codes, MSIL for C#/.NET and other instructions. While the present invention is for any intermediate language, Java will be referred to for examples and clarification.

[0012] By using an Accelerator Chip, systems with conventional processor chips and memory units can be accelerated for processing intermediate language instructions such as Java bytecodes. The Accelerator Chip is preferably placed in the path between the processor chip and the memory and can run intermediate language programs very efficiently. In a preferred embodiment, the Accelerator Chip includes a translator unit which translates at least some intermediate language instructions and an execution engine to execute the translated instructions. Execution of multiple intermediate languages can be supported in one accelerator concurrently or sequentially. For example, in one embodiment, the accelerator executes Java bytecodes as well as MSIL for C#/.NET.

[0013] Another embodiment of the present invention comprises an Accelerator Chip including a unit to execute intermediate language instructions, such as Java bytecodes and a memory interface. The memory interface is adapted to allow for memory access for the Accelerator Chip to at least one memory and to allow memory access to a separate processor chip to the at least one memory. By having an Accelerator Chip with such a memory interface, the Accelerator Chip can be placed in the path between the processor chip and memory unit.

[0014] Another embodiment of the present invention comprises an Accelerator Chip including a hardware translator unit, an execution engine, and a memory interface.

[0015] In another embodiment of the present invention, an intermediate language instruction cache operably connected to the hardware translator unit is used. By storing the intermediate language instructions in the cache, the execution speed of the programs can be significantly improved.

[0016] Another embodiment of the present invention comprises an Accelerator Chip including a hardware translator unit adapted to convert intermediate language instructions into native instructions, and a dedicated execution engine, the dedicated execution engine adapted to execute native instructions provided by the hardware translator unit. The dedicated execution engine only executing instructions provided by the hardware translator unit. The hardware translator unit rather than the execution engine preferably determines the address of the next intermediate language instructions to translate and provide to the dedicated execution engine. Alternatively the execution engine can determine the next address for the intermediate language instructions.

[0017] In one embodiment, the hardware translator unit only translates some intermediate language instructions, other intermediate language instructions cause a callback to the processor chip that runs a virtual machine to handle these exceptional instructions.

Brief Description of the Drawings

[0018] Fig. 1 is a diagram illustrating a system of one embodiment of the present invention.

[0019] Fig. 2 is a diagram illustrating an Accelerator Chip of one embodiment of the present invention.

[0020] Fig. 3 is a diagram of another embodiment of a system of the present invention.

[0021] Fig. 4A is a state machine diagram illustrating the modes of an Accelerator Chip of one embodiment of the present invention.

[0022] Fig. 4B is a state machine diagram illustrating modes of an accelerator chip of another embodiment of the present invention.

[0023] Fig. 5 is a table illustrating a power management scheme of one embodiment of an Accelerator Chip of the present invention.

[0024] Fig. 6 is a table illustrating one example of a list of bytecodes executed by an Accelerator Chip and a list of bytecodes that cause the callbacks to the processor chip for one embodiment of the system of the present invention.

[0025] Fig. 7 is a diagram that illustrates a common system memory organization for the memory units that can be used with one embodiment of the system of the present invention.

[0026] Fig. 8 is a table of pin functions for one embodiment of an Accelerator Chip of the present invention.

[0027] Fig. 9 is a diagram that illustrates memory wait states for different access times through the accelerator chip or without the accelerator chip for one embodiment of the present invention.

[0028] Fig. 10 is a high level diagram of an accelerator chip of one embodiment of the present invention.

[0029] Fig. 11 is a diagram of a system in which the accelerator chip interfaces with SRAMs.

[0030] Fig. 12 is a diagram of an accelerator chip in which the accelerator chip interfaces with SDRAMs.

[0031] Fig. 13 is a diagram of a system with an accelerator chip that has a larger bit interface to the memory than with the system on a chip.

[0032] Fig. 14 is a diagram of an accelerator chip including a graphics acceleration engine interconnected to an LCD display.

[0033] Fig. 15 is a diagram that illustrates the use of an accelerator chip within a chip stack package such that pins need not be dedicated for the interconnections to a flash memory and an SRAM.

[0034] Fig. 16A is a diagram of new instructions for one embodiment of the acceleration engine of one embodiment of the present invention.

[0035] Figs. 16B-16E illustrate the operation of the new instructions of Fig. 16A.

[0036] Fig. 17 is a diagram of one embodiment of an execution engine illustrating the logic elements for the new instructions of Fig. 16A.

[0037] Fig. 18A is a diagram that illustrates a Java bytecode instruction.

[0038] Fig. 18B illustrates a conventional microcode to implement the Java bytecode instruction.

[0039] Fig. 18C indicates the microcode with the new instructions of Fig. 16A to implement the Java bytecode instruction of Fig. 18A.

[0040] Fig. 19A illustrates the Java bytecode instruction LCMP.

[0041] Fig. 19B illustrates the conventional microcode for implementing the LCMP Java bytecode instruction of Fig. 19A.

[0042] Fig. 19C illustrates the microcode with the new instructions implementing the Java bytecode instruction LCMP of Fig. 19A.

Detailed Description of the Preferred Embodiment

[0043] Figure 1 illustrates a system 20 of one embodiment of the present invention. In this embodiment, an Accelerator Chip 22 is positioned between a processor chip 26 and memory units 24. Typically, a processor chip 26 interfaces with memory units 24. This is especially common in embedded systems used for communications, cell phones, personal digital assistants, and the like. In one embodiment the processor chip is a system on a chip (SOC) including a large variety of elements. For example, in one embodiment the processor chip 26 includes a direct memory access unit (DMA) 26a, a central processing unit (CPU) 26b, a digital signal processor unit (DSP) 26c and local memory 26d. In one embodiment, the SOC is a baseband processor for cellular phones for a wireless standard such as GSM, CDMA, W CDMA, GPRS, etc.

[0044] As will be described below, the Accelerator Chip 22 is preferably placed within the path between the processor chip 26 and memory units 24. The Accelerator Chip 22 runs at least portions of programs, such as Java, in an accelerated manner to improve the speed and reduce the power consumption of the entire system. In this embodiment, the Accelerator Chip 22 includes an execution unit 32 to execute intermediate language instructions, and a memory interface unit 30. The memory interface unit 30 allows the execution unit 32 on the Accelerator Chip 22 to access the intermediate language instructions and data to run the programs. Memory interface 30 also allows the processor chip 26 to obtain instructions and data from the memory units 24. The memory interface 30 allows the Accelerator Chip to be easily integrated with existing chip sets (SOC's). The accelerator function can be integrated as a whole or in part on the same chip stack package or on the same silicon with the SOC. Alternatively, it can be integrated into the memory as a chip stack package or on the same silicon.

[0045] The execution unit portions 32 of the Accelerator Chip 22 can be any type of intermediate language instruction execution unit. For example, in one embodiment a dedicated processor for the intermediate language instructions, such as a dedicated Java processor, is used.

[0046] In a preferred embodiment, however, the intermediate language instruction execution unit 32 comprises a hardware translator unit 34 which translates intermediate language instructions into translated instructions for an execution engine 36. The hardware translator unit 34 efficiently translates a number of intermediate language instructions. In one embodiment, the processor chip 26 handles certain intermediate language instructions which are not handled by the hardware translator unit. By having the translator unit efficiently translate some of the intermediate language instructions, then having these translated instructions executed by an execution engine, the speed of the system can be significantly increased. The translator can be microcode based, hence allowing the microcode to be swapped for Java versus C#/.NET.

[0047] Running a virtual machine completely in the processor 26 has a number of disadvantages. The translation portion of the virtual machine interpreter tends to be quite large and can be larger than the caches used in the processor chips. This causes the portions of the translating code to be repeatedly brought in and out of the cache from external memory, which slows the system. The translator unit 34 on the Accelerator Chip 22 does the translation without requiring translation software transfer from an external memory unit. This can significantly speed the operation of the intermediate language programs.

[0048] The use of callbacks for some intermediate language instructions is useful because it can reduce the size and power consumption of the Accelerator Chip 22. Rather than having a relatively complicated execution unit that can execute every intermediate language instruction, translating only certain intermediate language instructions in the translation unit 34 and executing them in the execution engine 36 reduces the size and power consumption of the Accelerator Chip 22. The intermediate language instructions executed by the accelerator are preferably the most commonly used instructions. The intermediate language instructions not executed by the accelerator chip can be implemented as callbacks such that they are executed on the SoC. Alternatively, the Accelerator Chip of one embodiment can execute every intermediate language instruction.

[0049] Also shown in the execution unit 32 of one embodiment is an interface unit and registers 42. In a preferred embodiment, the processor chip 26 runs a modified virtual machine which is used to give instructions to the Accelerator Chip 22. When a callback occurs, the translator unit 34 sets a register in unit 42 and the execution unit restores all the elements that need restoring and indicates such in the unit 42. In a preferred embodiment, the processor chip 26 has control over the Accelerator Chip 22 through the interface unit and registers 42. The execution unit 32 operates independently once the control is handed over to the Accelerator Chip.

[0050] In a preferred embodiment, an intermediate language instruction cache 38 is used associated with the translator unit 34. Use of an intermediate language instruction cache further speeds up the operation of the system and results in power savings because the intermediate language instructions need not be requested as often from the memory units 24. The intermediate language instructions that are frequently used are kept in the instruction cache 38. In a preferred embodiment, the instruction cache 38 is a two-way associative cache. Also associated with the system is a data cache 40 for storing data.

[0051] Although the translator unit is shown in Fig. 1 as separate from the execution engine, the translator unit can be incorporated into the execution engine. In that case, the central processing unit (CPU) or execution engine has a hardware translator subunit to translate intermediate language instructions into the native instructions operated on by the main portion of the CPU or the execution engine.

[0052] The intermediate language instructions are preferably Java bytecodes. Note that other intermediate language instructions, such as Multos bytecodes, MSIL, BREW, etc., can be used as well. For simplicity, the remainder of the specification describes an embodiment in which Java is used, but other intermediate language instructions can be used as well.

[0053] Figure 2 is a diagram of one embodiment of an Accelerator Chip. In this embodiment, the Java bytecodes are stored in the instruction cache 52. These bytecodes are then sent to the Java translator 34'. A bytecode buffer alignment unit 50 aligns the bytecodes and provides them to the bytecode decode unit 52. In a preferred embodiment, for some bytecodes, instruction level parallelism is done with the bytecode decode unit 52 combining more than one Java bytecode into a single translated instruction. In other situations, the Java bytecode results in more than one native instruction as required. The Java bytecode decode unit 52 produces indications which are used by the instruction composition unit 54 to produce translated instructions. In a preferred embodiment, a microcode lookup table unit associated with or within unit 54 produces the base portion of the translated instructions with other portions provided from the Stack and Variable Managers 56 which keep track of the meaning of the locations in the register file 58 of the processor 60 in execution engine 36'. In one embodiment, the register file 58 of the processor 60 stores the top eight Java operand stack values, sixteen Java variable values and four scratch values.

[0054] In a preferred embodiment, the execution engine 36' is dedicated to only execute the translated instructions from the Java translating unit. In a preferred embodiment, processor 60 is a reduced instruction set computing (RISC) processor or a DSP, or VLIW or CISC processor. These processors can be customized or modified so its instruction set is designed to efficiently execute the translated instructions. Instructions and features that are not needed are preferably removed from the instruction set of the execution engine to produce a simpler execution engine - for example, interrupts are preferably not used. Furthermore, the execution engine 36' need not directly calculate the location of the next instruction to execute. The Java translator unit 34' can instead calculate the addresses of the next Java bytecode to translate. The processor 60 produces flags to controller 62 which then calculates the location of the next Java bytecode to translate. Alternatively, standard processors can be used.

[0055] In one embodiment, the bytecode buffer control unit 72 checks how many bytecode bytes are accepted into the Java translator, and modifies the Java program counter 70. The controller 62 can also modify the Java program counter. The address unit 64 obtains the next instruction either from the instruction cache or from external memory. Note that, for example, the controller 62 can also clear out the Java translator unit's pipeline if required by a "branch taken" or a callback. Data from the processor 60 is also stored in the data cache 68.

[0056] When the virtual machine modifies the bytecode to the quick form, the cache line in the hardware accelerator holding the bytecode being modified needs to be invalidated. The same is true when the virtual machine reverses this process and restores the bytecode to the original form. Additionally, the callbacks invalidate the appropriate cache line in the instruction cache using a cache invalidate register in the interface register.

[0057] In some embodiments, when quick bytecodes are used, the modified instructions are stored back into the instruction cache 52. When quick bytecodes are used, the system must keep track of how the Java bytecodes are modified and eventually have instruction consistency between the cache and the external memory.

[0058] In one embodiment, the decoded bytecodes from the bytecode decode unit are sent to a state machine unit and Arithmetic Logic Unit (ALU) in the instruction composition unit 54. The ALU is provided to rearrange the bytecode instructions to make them easier to be operated on by the state machine and perform various arithmetic functions including computing memory references. The state machine converts the bytecodes into native instructions using the lookup table. Thus, the state machine provides an address which indicates the location of the desired native instruction in the microcode look-up table. Counters are maintained to keep a count of how many entries have been placed on the operand stack, as well as to keep track of and update the top of the operand stack in memory and in the register file. In a preferred embodiment, the output of the microcode look-up table is augmented with indications of the registers to be operated on in the register file. The register indications are from the counters and interpreted from bytecodes. To accomplish this, it is necessary to have a hardware indication of which operands and variables are in which entries in the register file. Native Instructions are composed on this basis. Alternately, these register indications can be sent directly to the register file.

[0059] In another embodiment of the present invention, the Stack and Variable manager assigns Stack and Variable values to different registers in the register file. An advantage of this alternate embodiment is that in some cases the Stack and Var values may switch due to an Invoke Call and such a switch can be more efficiently done in the Stack and Var manager rather than producing a number of native instructions to implement this.

[0060] In one embodiment, a number of important values can be stored in the hardware accelerator to aid in the operation of the system. These values stored in the hardware accelerator help improve the operation of the system, especially when the register files of the execution engine are used to store portions of the Java stack.

[0061] The hardware translator unit preferably stores an indication of the top of the stack value. This top of the stack value aids in the loading of stack values from the memory. The top of the stack value is updated as instructions are converted from stack-based instructions to register-based instructions. When instruction level parallelism is used, each stack-based instruction which is part of a single register-based instruction needs to be evaluated for its effects on the Java stack.

[0062] In one embodiment, an operand stack depth value is maintained in the hardware accelerator. This operand stack depth indicates the dynamic depth of the operand stack in the execution engine register files. Thus, if eight stack values are stored in the register files, the stack depth indicator will read "8." Knowing the depth of the stack in the register file helps in the loading and storing of stack values in and out of the register files.

[0063] Additionally, a frame stack can be maintained in the hardware with its own underflow/overflow and frame depth indication to indicate how many frames are on the frame stack. The frame stack can be a stand-alone stack or incorporated within the CPU's register file. In a preferred embodiment, the frame stack and the operand stack can be within the same register file of the CPU. In another embodiment, the frame stack and the operand stack are different entities. The local variables would also be stored in a separate area of the CPU register file which also has the operand stack and/or the frame stack.

[0064] In a preferred embodiment, a minimum stack depth value and a maximum stack depth value are maintained by the hardware translator unit. The stack depth value is compared to the required maximum and minimum stack depths. When the stack value goes below the minimum value, the hardware translator unit composes load instructions to load stack values from the memory into the register file. When the stack depth goes above the maximum value, the hardware translator unit composes store instructions to store stack values back out to the memory.

[0065] In one embodiment, at least the top eight (8) entries of the operand stack in the execution engine register file operate as a ring buffer, and the ring buffer is maintained in the accelerator and is operably connected to a overflow/underflow unit.

[0066] The hardware translator unit also preferably stores an indication of the operands and variables stored in the register file of the execution engine. These indications allow the hardware accelerator to compose the converted register-based or native instructions from the incoming stack-based instructions.

[0067] The hardware translator unit also preferably stores an indication of the variable base and operand base in the memory. This allows for the composing of instructions to load and store variables and operands between the register file of the execution engine and the memory. For example, when a variable (Var) is not available in the register file, the hardware issues load instructions. The hardware is adapted to multiply the Var number by four and adding the Var base to produce the memory location of the Var. The instruction produced is based on knowledge that the Var base is in a temporary native execution engine register. The Var number times four can be made available as the immediate field of the native instruction being composed, which may be a memory access instruction with the address being the content of the temporary register holding a pointer to the Vars base plus an immediate offset. Alternatively, the final memory location of the Var may be read by the execution engine as an instruction and then the Var can be loaded.

[0068] In one embodiment, the hardware translator unit marks the variables as modified when updated by the execution of Java bytecodes. The hardware accelerator can copy variables marked as modified to the system memory for some bytecodes.

[0069] In one embodiment, the hardware translator unit composes native instructions wherein the native instruction's operands contain at least two native execution engine register file references where the register file contents are the data for the operand stack and variables.

[0070] In one embodiment a stack-and-variable-register manager maintains indications of what is stored in the variable and stack registers of the register file of the execution engine. This information is then provided to the decode stage and microcode stage in order to help in the decoding of the Java bytecode and generating appropriate native instructions.

[0071] In a preferred embodiment, one of the functions of a Stack-and-Var register manager is to maintain an indication of the top of the stack. Thus, if for example registers R1-R4 store the top 4 stack values from memory or by executing bytecodes, the top of the stack will change as data is loaded into and out of the register file. Thus, register R2 can be the top of the stack and register R1 be the bottom of the stack in the register file. When a new data is loaded into the stack within the register file, the data will be loaded into register R3, which then becomes the new top of the stack, the bottom of the stack remains R1. With two more items loaded on the stack in the register file, the new top of stack in the register file will be R1 but first R1 will be written back to memory by the accelerator's overflow/underflow unit, and R2 will be the bottom of the partial stack in the register file.

[0072] Fig. 3 shows the main functional units within an example of an accelerator chip accelerator as well as how it interfaces into a typical wireless handset design. The accelerator chip integrates between the host microprocessor (or the SOC that includes an embedded microprocessor) and the system SRAM and/or Flash memory. From the perspective of the host microprocessor and system software, the system SRAM and/or Flash memory is behind the accelerator chip.

[0073] The Accelerator Chip has direct access to the system SRAM and/or Flash memory. The host microprocessor (or microprocessor within an SOC) has transparent access to the system SRAM or Flash memory through the Accelerator Chip ("the system memory is behind the accelerator").

[0074] The Accelerator Chip preferably synchronizes with the host microprocessor via a monitor within its companion software kernel. The Software Kernel (or the processor chip) loads specific registers in the accelerator chip with the address of where Java bytecode instructions are located, and then transfers control to the accelerator chip to begin executing. The software kernel then waits in a polling loop running on the host microprocessor reading the run mode status until either it detects that it is necessary to process a bytecode using the callback mechanism or until all bytecodes have been executed. The polling loop can be implemented by reading the "run mode" pin electrically connected between the accelerator chip and a general purpose I/O pin on the SOC. Alternatively, the same status of the "run mode" can be polled by reading the registers within the accelerator chip. In either of these cases, the accelerator chip automatically enters its power-saving sleep state until callback processing has completed or it is directed to execute more bytecodes.

[0075] The Accelerator Chip fetches the entire Java bytecode including the operands from memory, through its internal caches, and executes the instruction. Instructions and data resident in the caches are executed faster and at reduced power consumption because system memory transactions are avoided. Bytecode streams are buffered and analyzed prior to being interpreted using an optimizer based on instruction level parallelism (ILP). The ILP optimizer coupled with locally cached Java data results in the fastest execution possible for each cycle.

[0076] Since the Accelerator Chip is a separate stand-alone Java bytecode execution engine, it processes concurrently while the host microprocessor is either waiting in its polling loop or processing interrupts. Furthermore, the Accelerator Chip is only halted during instances when the host microprocessor needs to access system memory behind it, and the accelerator chip also wants to access system memory at the same time. For example, if the host microprocessor is executing an interrupt service routine or other software from within its own cache, then the Accelerator Chip can concurrently execute bytecodes. Similarly, if Java bytecode instructions and data reside within the Accelerator Chip's internal caches, then the accelerator can concurrently execute bytecodes even if the host microprocessor needs to access system memory behind it.

[0077] Fig. 4A is a state machine showing the two primary modes of the accelerator chip of one embodiment: sleep and running (executing Java bytecode instructions). The accelerator chip automatically transitions between its running and sleep states. In its sleep state, the accelerator chip draws minimal power because the Java engine core and associated components are idled.

[0078] Fig. 4B is a diagram of the states of the accelerator chip of another embodiment of the system of the present invention, further including a standby mode. The standby mode is used during callbacks. In order to reduce power, only the clocks to the Java registers are on. In the standby mode, the processor chip is running the virtual machine to handle the Java bytecode that causes the callback. Since the accelerator chip is in the standby mode, it can quickly recover without having to reset all of the Java registers.

[0079] Fig. 5 shows what components are active and idle in each mode of the state machine of Fig. 4A. When the JVM is not running or when the system determines that additional power savings are appropriate, the Accelerator Chip automatically assumes its sleep mode.

[0080] Once activated, the Accelerator Chip runs until any of the following events occurs:

1. When it is necessary that a Java bytecode instruction be executed by the host microprocessor via the software callback mechanism.
2. The host microprocessor needs to access system memory, which typically only occurs during interrupt and exception processing.
3. The host microprocessor halts the accelerator chip by forcing it into its sleep mode.

[0081] The Accelerator Chip is disabled (in its sleep mode) and transparent to all native resident software by default, and it is enabled when a modified Java virtual machine initializes it and calls on it to execute Java bytecode instructions. When the accelerator chip is in its sleep mode, accesses to SRAM or Flash memory from the host microprocessor simply pass through the Accelerator chip.

[0082] The Accelerator Chip includes a memory controller as an integral part of its memory interface circuitry that needs to be programmed in a manner typical of SRAM and/or Flash memory controllers.. The actual programming is done within the software kernel with the specific memory addresses set according to each device's unique architecture and memory map. As part of the modified Java virtual machine's initialization sequence, registers within accelerator chip are loaded with the appropriate information. When the system calls on its JVM to execute Java software, it first loads the address of the start of the Java bytecodes into the Java Program Counter (JP) of the Accelerator Chip. The kernel then begins running on the host microprocessor monitoring the Accelerator Chip for when it signals that it has completed executing Java bytecodes. Upon completion the Accelerator Chip goes into its sleep mode and its kernel returns control to the JVM and the system software.

[0083] The Accelerator chip does not disturb interrupt or exception processing, nor does it impose any latency. When an interrupt or exception occurs while the Accelerator Chip is processing, the host microprocessor diverts to an appropriate handler routine without affecting accelerator chip. Upon return from the handler, the host microprocessor returns execution to the software kernel and in turn resumes monitoring the Accelerator Chip. Even when the host microprocessor takes over the memory bus, the Accelerator Chip can continue executing Java bytecodes from its internal cache, which can continue so long as a system memory bus conflict does not arise. If a conflict arises, a stall signal can be asserted to halt the accelerator.

[0084] The Accelerator Chip has several shared registers that are located in its memory map at a fixed offset from a programmable base. The registers control its operation and are not meant for general use, but rather are handled by code within the Software Kernel.

[0085] Referring to Fig. 3, it can be seen that the Accelerator Chip is positioned between the host microprocessor (or the SOC that includes an embedded microprocessor) and the system SRAM and/or Flash memory. All system memory accesses by the host microprocessor therefore pass through the Accelerator Chip. In one embodiment, while fully transparent to all system software, a latency of approximately 4 nanoseconds is introduced for each direction, contributing to a total latency of approximately 8 nanoseconds for each system memory transaction.

[0086] Fig. 6 is a table that illustrates one embodiment of a list of Java bytecodes that are executed by the Java execution unit on the Accelerator Chip and a list of bytecodes that cause a callback to the modified JVM running on the processor chip. Note that the most common bytecodes are executed on the Accelerator Chip. Other less common and more complex bytecodes are executed in software on the processor chip. By excluding certain Java bytecodes from the Accelerator Chip, the Accelerator Chip complexity and power consumption can be reduced.

[0087] Fig. 7 illustrates a typical memory organization and the types of software and data that can be stored in each type of memory. Placement of the items listed in the table below allows the accelerator chip to access the bytecodes and corresponding data items necessary for it to execute Java bytecode instructions.

[0088] The operating system running on the host microprocessor is preferably set up such that virtual memory equals real memory for all areas of memory that the accelerator chip will access as part of its Java processing.

[0089] Integration with a Java virtual machine is preferably accomplished through the modifications as listed below.

1. Insertion of modified initialization code into the JVM's own initialization sequence.
2. Removal of the Java bytecode interpreter and installing the modified software kernel. This includes redirecting the functionality for the Java bytecode instructions that are not directly executed within the accelerator chip hardware into the callback mechanism enabled by the accelerator chip software kernel. Additionally, for quick bytecodes, when the JVM modifies the bytecode to its quick form, the cache line within the Hardware Accelerator instruction cache holding the bytecode being modified ("quickified") must be invalidated. The same is true when JVM reverses this process and restores the bytecode to its original form. The accelerator chip and its software kernel preferably provide Application Programming Interface (API) calls to handle these situations.
3. Adapting the garbage collector. The JVM's garbage collector invalidates the data cache within the accelerator chip before scanning the Java Heap or Java Stack to avoid cache coherency problems. This is preferably accomplished using an API function within the Software Kernel.

[0091] One embodiment of the Accelerator Chip preferably interfaces with any system that has been designed with asynchronous SRAM and/or asynchronous Flash memory including page mode Flash memory. In such circumstances, the accelerator chip easily integrates because it looks to the system like an SRAM or Flash device. No other accommodations are necessary for integration. The Accelerator Chip has its own memory controller and correspondingly the ability to access memory "behind the accelerator" directly via an internal program counter (IPC). As with any program counter, the JP points to the address of the next instruction to be fetched and executed. This allows the accelerator chip to operate asynchronously and concurrently with regard to the host microprocessor.

[0092] Fig. 8 is a table that illustrates on example of the accelerator pin functions for one example of an Accelerator Chip of the present invention.

[0093] In a preferred embodiment, the pins going to the processor chip and going to the memory are located near each other in order to keep the delay through the chip at the minimum for the bypass mode.

[0094] Fig. 9 is a diagram that illustrates the wait states for different access times and bus speeds with an embodiment of a hardware accelerator positioned in between the processor chip, such as an SOC, and the memory. Note that in some cases, additional wait states for access times need to be added due to the introduction of the hardware accelerator in the path between the processor chip and the memory.

[0095] Fig. 10 is a diagram of a hardware accelerator of one embodiment of the present invention. The hardware accelerator 100 includes bypass logic 102. This connects to the system on a chip interface 104 and memory interface 106. The memory controller 108 is interconnected to the interface register 110 which is used to send messages between the system on the chip and the hardware accelerator. Instructions going through the memory controller 108 to the instruction cache 112 and the data from the data cache 114 are sent to the memory controller 108. The intermediate language instructions from an instruction cache 112 are sent to the hardware translator 114, which translates them to native instructions, and sends the translated instructions to the execution engine 116. In this embodiment, the execution engine 116 is broken down into a register read stage 116A, an execution stage 116B and a data cache stage 116C.

[0096] Fig. 11 is a diagram of a hardware accelerator 120 which is used to interface with SRAM memories. Since SRAM memories and SDRAM memories can be significantly different, in one embodiment, there is a dedicated hardware accelerator for each type of memory. Fig. 11 shows the hardware accelerator including an instruction cache, hardware translator, data cache, execution engine, a phase lock loop (PLL) circuit which is used to set the internal clock of the hardware accelerator such that it is synched to an external clock, the interface registers and SRAM slave interface and SRAM master interface. The SRAM slave interface interconnecting to the system on a chip, and SRAM master interface interconnecting to the memory. The diagram of Fig. 11 emphasizes the fact that the connections between the system on a chip and the memory are separate and dealt with separate interfaces. Thus, interactions between the hardware accelerator and the system on a chip and interactions between the hardware accelerator and the memory can be done concurrently for independent operations. Shown interconnected between the system on a chip and the hardware accelerator are address lines, data lines, byte select lines, write enable lines, read enable lines, chip select lines and the like. Note that the asynchronous flash pins can go directly between the processor chip and the asynchronous flash unit. The hardware accelerator chip can modify the chip selection memory addressing capabilities of the system on a chip. In one embodiment, an optional system on a chip memory is stored in the SRAM slave interface. The host processor enters a wait loop to check the run mode set by the interface register of the hardware accelerator. The system on a chip obtains the register loop check program from the SRAM slave interface. The hardware accelerator 120 is not interrupted by the SOC accessing the loop program in the external memory and, thus, can more efficiently run the intermediate language programs stored in the external memory. Note that the hardware accelerator 120 can include a JTAG test unit.

[0097] Fig. 12 illustrates an embodiment of the system of the present invention in which the hardware accelerator 130 includes an SDRAM slave and SDRAM master interfaces. The control lines for interconnecting to an SDRAM are significantly different from the control lines interconnecting to an SRAM so that it makes sense to have two different versions of the hardware accelerator in one embodiment. Additional lines for the SDRAM include a row select, column select and write enable lines.

[0098] Fig. 13 illustrates a diagram of a host hardware accelerator 140. This embodiment has a 16-bit interconnection from the processor chip and a 32-bit connection between the hardware accelerator 140 and the memory. The interconnection between the memory and the hardware accelerator will operate faster than the interconnection between the processor and the memory. A host burst buffer is included in the host accelerator 140 such that data can be buffered between the processor chip and the memory.

[0099] Fig. 14 illustrates an embodiment in which the hardware accelerator 150 includes a graphics accelerator engine 152 and an LCD controller and display buffers 154. This allows the hardware accelerator 150 to interact with the LCD display 156 in a direct manner. The Java standards include a number of libraries. These libraries are typically implemented such that devices can run a different type of code other than Java code to implement them. One new type of library includes graphics for LCD display. For example, a canvas application is used for writing applications that need to handle low-level events and issue graphical calls for drawing on the LCD display. Such an application would typically be used for games and the like. In the embodiment of Fig. 14, a graphics accelerator engine 152 and LCD control and display buffer engines 154 are placed in the hardware accelerator 150, so the control of the system need not be passed to the processor chip. Whenever a graphics element is to be run, a Java program rather than the conventional program is used. The Java program stored in the memory is used to update the LCD display 156. In one embodiment, the Java program uses a special identifier bytecode which is used by the hardware accelerator 150 to determine that the program is for LCD graphics acceleration engine 152. It is not always necessary to have the LCD controller on the same chip if the function is available on the SOC. In this case, only the graphics would still be on the accelerator. The graphics can be for 2D as well as 3D graphics. Additionally, a video camera interface can also be included on the chip. The camera interface unit would interface to a video unit where the video image size can be scaled and/or color space conversion can be applied. By setting certain registers within the accelerator chip it is possible to merge video and graphics to provide certain blend and window effects on the display. The graphics unit would have its own frame buffer and optionally a Z-buffer for 3D. For efficiency, it would be optimal to have the graphics frame buffer in the accelerator chip and have the Z-buffer in the system SRAM or system SDRAM.

[00100] Fig. 15 is a diagram of a chip stack package 160 which includes an accelerator chip 162, flash chip 164 and SRAM chip 166. By putting the accelerator chip 162 in a package along with the memory chips 164 and 166, the number of pins that need to be dedicated on the package for interconnecting between the accelerator chip and the memory can be reduced. In the example of Fig. 15, the reduction in the number of pins allows a set of pins to be used for a bus data and addresses to an auxiliary memory location. Positioning the accelerator chip on the same package as the flash memory chip and SRAM chip also reduces the memory access time for the system.

[00101] Figs. 16-19 are diagrams that illustrate new instructions which are useful for adding to the accelerator engine of one embodiment of the present invention, so that it efficiently executes translated intermediate language instructions, especially Java bytecodes. The embodiment of Figs. 16-19 can be used within a hardware accelerator chip, but can also be used with other systems using a hardware translator and an execution engine.

[00102] Fig. 16A illustrates new instructions for an execution engine that speeds up the operation of translated instructions. By having these translated instructions, the operation of the execution engine running the translated instructions can be improved. The instructions SGTLT0 and SGTLT0U use the C, N and Z outputs of the adder/subtractor of a previous operation in order to then write a -1, 0 or 1 in a register. These operations improve the efficiency of the Java bytecode LCMP. The bounds check operation (BNDCK) and the load and store index instructions with the register null check speed the operation of the translated instructions for the Java bytecodes that do indexed array access.

[00103] Fig. 16B illustrates the operation of the instruction SGTLT0. When the last subtract or add produces a Z bit of 1, the output into the register is a 0. When the previous Z bit is a 0, and the N bit is a 0, the output into the register is a 1. When the Z bit is 0, and the N bit is a 1, the output into the register is a -1.

[00104] Fig. 16C illustrates the instruction SGTLT0U, in which an unsigned operation is used. In this example, if the Z value is high, the output to the register is a 0. If the Z value is low, and the carry is a 0, the output to the register is -1. If the Z value is low, and the carry is 1, the output to the register is 1.

[00105] Fig. 16D illustrates the bound check instruction BNDCK. In this instruction, the index is subtracted from the array size value. If the index is greater than the array size, the carry will be 1, and an exception will be created. If the index is less than the array size, the carry will be 0, and no exception will be produced.

[00106] Fig. 16E shows indexed instructions, including the index loads and index stores that check a register for a null value, in addition to the index operation. In this case, if the array pointer register is a 0, an exception occurs. If the array pointer is not a 0, no exception occurs.

[00107] Fig. 17 illustrates one example of an execution engine implementing some of the details of the system for the new instructions of Fig. 16A. For the indexed loads, the zero checking logic 170 checks to see whether the value of the index stored in a register, such as register H is 0. When the zero check enable is set (meaning that the instruction is one of the four instructions LDXNC, LWXNC, STXNC, or SWXNC), the zero check enable is set high. Note that the other operations for the load can be done concurrently with this operation. The zero checking logic 170 ensures that the pointer to the array is not 0, which would indicate a null value for the array pointer. When the pointer is correctly initialized, the value will not be a 0 and thus, when the value is a 0, an exception is created.

[00108] The adder/subtractor unit 172 produces a result and also produces the N, Z and C bits which are sent to the N, Z and C logic 174. For the bounds checking case, the bounds checking logic 176 checks to see whether the index is inside the size of the array. In the bounds checking, the index value is subtracted from the array size, the index value will be stored in one register, while the array value is stored in another register. If there is a carry, this indicates an exception, and the bounds check logic 176 produces an index out of range exception when the bounds checking is enabled.

[00109] Logical unit 178 includes the new logic 180. This new logic 180 implements the SGTLT0 and SGTLT0U instructions. Logic 180 uses the N and Z carry bits from a previous subtraction or add. As illustrated by Figs. 16A and 16C, the logic 160 produces a 1, 0 or -1 value, which is then sent to the multiplexer (mux) 182. When the SGTLT0 or SGTLTU instructions are used, the value from the logic 180 is selected by the mux 182.

[00110] Fig. 18A illustrates the Java bytecode instruction IALOAD. The top two entries of the stack are an index and an array reference, which are converted to a single value indicated by the index offset into the array. With the conventional instructions as shown in Fig. 18B, the array reference needs to be compared to 0 to see whether a null pointer exception is to be produced. Next, a branch check is done to determine whether the index is outside of array bounds. The index value address is calculated and then loaded. In Fig. 18C, with the new instructions, the LWXNC reference does a zero check for the register containing the array pointer. The bounds check operation makes sure the index is within the array size. Thereafter the add to determine the address and the load is done.

[00111] Fig. 19A illustrates the operation of an LCMP instruction, in which the top two values of the stack include two words for the first value. The second two values on the stack contain the value 1 word 1 and 2, and an integer result is produced based on whether value 1 is equal to value 2, value 1 is greater than value 2 or value 1 is less than value 2.

[00112] Fig. 19B illustrates a conventional instruction implementation of the Java LCMP instruction. Note that a large number of branches with the required time is needed.

[00113] In Fig. 19C, the existence of the SGLT0U instruction simplifies the operation of the code and can speed the system of the present invention.

[00114] The hardware translator is enabled to translate into the above new instructions. This makes the translation from Java bytecodes more efficient.

[00115] The Accelerator Chip of the present invention has a number of advantages. The Accelerator Chip directly accesses system memory to execute Java bytecode instructions while the host microprocessor services its interrupts, contributing to speed-up of Java software execution. Because the accelerator chip executes bytecodes and does not compile them, it does not impose additional memory requirements, making it a less costly and more efficient solution than using ahead-of-time (AOT) or just-in-time (JIT) compilation techniques. System level energy usage is minimized through a combination of faster execution time, reduced memory accesses and power management integrated within the accelerator chip. When not executing bytecodes, the Accelerator Chip is automatically in its power-saving sleep mode. The accelerator chip uses data localization and instruction level parallelism (ILP) optimizations achieve maximum performance. Data held locally within the accelerator chip preferably includes top entries on the Java stack and local variables that increase the effectiveness of the ILP optimizations and reduce accesses to system memory. These techniques result in fast and consistent execution and reduced system energy usage. This is in contrast to typical commercial microprocessors that rely on software interpretation that treat bytecodes as data and therefore derive little to no benefit from their instruction cache. Also, because Java bytecodes along with their associated operands vary in length a typical software bytecode interpreter must perform several data accesses from memory to complete each Java bytecode fetch cycle - a process that is inefficient in terms of performance and power consumption. The Java Virtual Machine (JVM) is a stack-based machine and most software interpreters locate the entire Java stack

in system memory requiring several costly memory transactions to execute each Java bytecode instruction. As with bytecode fetches, the memory transactions required to manage and interact with a memory based Java stack are costly in terms of performance and increased system power consumption.

[00116] The Accelerator Chip easily interfaces directly to typical memory system designs and is fully transparent to all system software providing its benefits without requiring any porting or new development tools. Although the JVM is preferably modified to drive Java bytecode execution into the accelerator chip, all other system components and software are unaware of its presence. This allows any and all commercial development tools, operating systems and native application software to run as-is without any changes and without requiring any new tools or software. This also preserves the investment in operating system software, resident applications, debuggers, simulators or other development tools. Introduction of a accelerator chip is also transparent to memory accesses between the host microprocessor and the system memory but may introduce wait states. The Accelerator Chip is useful for mobile/wireless handsets, PDAs and other types of Internet Appliances where performance, device size, component cost, power consumption, ease of integration and time to market are critical design considerations.

[00117] In one embodiment, the accelerator chip is integrated as a chip stack with the processor chip. In another embodiment, the accelerator chip is on the same silicon as the memory. Alternatively, the accelerator chip is integrated as a chip stack with the memory. In a further embodiment, the processor chip is a system on a chip. In an alternative embodiment, the system on a chip is adapted for use in cellular phones.

[00118] In one embodiment, the accelerator chip supports execution of two or more intermediate languages, such as Java bytecodes and MSIL for C#/.NET.

[00119] In one embodiment of the present invention, the system comprises at least one memory, a processor chip operably connected to the at least one memory, and an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate

language, the hardware accelerator including a accelerator of a Java processor for the execution of intermediate language instructions.

[00120] In a further embodiment of the present invention, the system comprises at least one memory, a processor chip operably connected to the at least one memory, and an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language, wherein some instructions generate a callback and get executed on the processor chip.

[00121] While the present invention has been described with reference to the above embodiments, this description of the preferred embodiments and methods is not meant to be construed in a limiting sense. For example, the term Java in the specification or claims should be construed to cover successor programming languages or other programming languages using basic Java concepts (the use of generic instructions, such as bytecodes, to indicate the operation of a virtual machine). It should also be understood that all aspects of the present invention are not to be limited to the specific descriptions, or to configurations set forth herein. Some modifications in form and detail the various embodiments of the disclosed invention, as well as other variations in the present invention, will be apparent to a person skilled in the art upon reference to the present disclosure. It is therefore contemplated that the following claims will cover any such modifications or variations of the described embodiment as falling within the true spirit and scope of the present invention.

WHAT IS CLAIMED IS:

1. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language.

2. The system of Claim 1 wherein the programs in an intermediate language instructions are Java bytecodes.

3. The system of Claim 2 wherein the processor runs a modified Java virtual machine.

4. The system of Claim 1 wherein the intermediate language is in bytecode form.

5. The system of Claim 1 wherein the accelerator chip is positioned on a memory bus.

6. The system of Claim 1 wherein the memory comprises a number of memory units.

7. The system of Claim 6 wherein the memory units include a static random access memory.

8. The system of Claim 6 wherein the memory units include a flash memory.

9. The system of Claim 1 wherein the processor runs a modified virtual machine.

10. The system of Claim 1 wherein the accelerator chip does not run certain bytecodes but instead has a callback to the virtual machine running on the processor chip.

11. The system of Claim 1 wherein the accelerator chip has a sleep mode with low power consumption.

12. The system of Claim 1 wherein the processor chip is a system on a chip.

13. The system of Claim 1 wherein the accelerator chip includes a hardware translator unit adapted to convert intermediate language instructions into native instructions and an execution unit adapted to execute the native instructions provided by the hardware translator unit.

14. The system of Claim 13 wherein the hardware translator unit is adapted to convert Java bytecodes into native instructions.

15. The system of Claim 1 wherein the accelerator chip includes an interface adapted to allow memory access for the accelerator chip to at least one memory, and to allow for access for the processor chip to the at least one memory.

16. The system of Claim 15 wherein the interface comprises a first interface to the processor chip and a second interface to the memory unit, the second and first interface adapted to operate independently.

17. The system of Claim 1 wherein the accelerator chip includes an instruction cache operably connected to store instructions to be executed within the accelerator chip.

18. The system of Claim 17 wherein the accelerator chip includes an instruction cache operably connected to store instructions to be executed within the accelerator chip.

19. The system of Claim 1 wherein the accelerator chip includes a hardware translator unit and a dedicated execution unit adapted to execute native instructions provided by the hardware translator unit, the dedicated execution engine only executing instructions provided by the hardware translator unit.

20. The system of Claim 1 wherein the accelerator chip is integrated as a chip stack with the processor chip.

21. The system of Claim 1 wherein the accelerator chip is on the same silicon as the memory.

22. The system of Claim 1 wherein the accelerator chip is integrated as a chip stack with the memory.

23. The system of Claim 1 wherein the processor chip is a system on a chip.

24. The system of Claim 23 wherein the system on a chip is adapted for use in cellular phones.

25. The system of Claim 1 wherein the accelerator chip supports execution of two or more intermediate languages.

26. The system of Claim 25 wherein the intermediate languages are Java bytecodes and MSIL for C#/.NET.

27. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

a accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language, the hardware accelerator including a hardware translator unit adapted to covert intermediate language instructions into native instructions, and an execution engine adapted to execute the native instructions provided by the hardware translator unit.

28. The system of Claim 27 wherein the programs in an intermediate instruction language are Java programs and the hardware translator unit coverts Java bytecodes into native instructions.

29. The system of Claim 28 wherein the processor runs a modified Java virtual machine.

30. The system of Claim 27 wherein the accelerator chip is positioned on a memory bus in between the processor chip and the at least one memory.

31. The system of Claim 27 wherein the memory comprises a number of memory units.

32. The system of Claim 31 wherein one of the memory units comprises a static random access memory.

33. The system of Claim 31 wherein at least one of the memory units comprises a flash memory.

34. The system of Claim 27 wherein the processor runs a modified virtual machine.

35. The system of Claim 34 wherein the accelerator chip does not execute certain intermediate language instructions and a callback occurs when these intermediate language instructions occur, these intermediate language instructions being executed on the modified virtual machine running on the processor chip.

36. The system of Claim 27 wherein the accelerator chip has a sleep mode with low power consumption.

37. The system of Claim 27 wherein the processor chip is a system on a chip.

38. The system of Claim 27 wherein the accelerator chip includes an interface adapted to allow for memory access for the accelerator chip to at least one memory, and to allow for memory access for the processor chip to the at least one memory.

39. The system of Claim 27 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing the intermediate language instructions to be converted.

40. The system of Claim 27 wherein the execution engine is a dedicated execution engine only executing instructions provided by the hardware translator unit.

41. An accelerator chip comprising:

a unit adapted to execute intermediate language instructions; and

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory.

42. The accelerator chip of Claim 41 wherein the intermediate language instructions are Java bytecodes.

43. The accelerator chip of Claim 41 wherein the accelerator chip does not execute certain intermediate language instructions but instead causes a callback to the separate processor chip.

44. The accelerator chip of Claim 41 wherein the accelerator chip has a sleep mode with low power consumption.

45. The accelerator chip of Claim 41 wherein the accelerator chip includes an instruction cache operably connected to operably connected to the hardware translator unit storing intermediate language instructions to be converted.

46. The accelerator chip of Claim 41 wherein the unit includes a hardware translator unit adapted to convert intermediate language instructions into native instructions and an execution engine adapted to execute native instructions provided by the hardware translator unit.

47. The accelerator chip of Claim 41 wherein the unit comprises a dedicated processor whose native instruction is the intermediate language instruction.

48. An accelerator chip comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions;

an execution engine adapted to execute the native instructions provided by the hardware translator unit; and

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory.

49. The accelerator chip of Claim 48 wherein the intermediate language instructions are Java bytecodes.

50. The accelerator chip of Claim 48 wherein the accelerator chip does not execute every intermediate language instruction but some intermediate language instructions cause a callback to the separate processor chip running a modified virtual machine.

51. The accelerator chip of Claim 48 wherein the accelerator chip has a sleep mode with low power consumption.

52. The accelerator chip of Claim 48 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted.

53. The accelerator chip of Claim 48 wherein the execution engine is a dedicated execution engine only executing instructions provided by the hardware translator unit.

54. An accelerator chip comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions;

an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted;

an execution engine adapted to execute the native instructions provided by the hardware translator unit; and

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory.

55. The accelerator chip of Claim 47 wherein the intermediate language instructions are Java bytecodes.

56. The accelerator chip of Claim 54 wherein the accelerator chip does not execute every intermediate language instruction but for some intermediate language instructions causes a callback to a processor running a modified virtual machine.

57. The accelerator chip of Claim 54 wherein the accelerator chip has a sleep mode with low power consumption.

58. The accelerator chip of Claim 54 wherein the execution engine is a dedicated execution engine adapted to only execute instructions provided by the hardware translator unit.

59. An accelerator chip comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

a dedicated execution engine adapted to execute the native instructions provided by the hardware translator unit, the dedicated execution engine only executing instructions provided by the hardware translator unit, wherein the hardware translator unit, rather than the execution engine, determines the address of the next intermediate language instruction to translate and provide to the dedicated execution engine.

60. The accelerator chip of Claim 59 wherein the intermediate language instructions are Java bytecodes.

61. The accelerator chip of Claim 59 wherein the accelerator chip does not execute every intermediate language instruction but some intermediate language instructions cause a callback to a separate processor chip running a modified virtual machine for interpretation.

62. The accelerator chip of Claim 59 wherein the accelerator chip includes a sleep mode with low power consumption.

63. The accelerator chip of Claim 59 wherein the accelerator chip includes an interface adapted to allow for memory access for the accelerator chip to at least one memory and allow for memory access for a separate processor chip to the at least one memory.

64. The accelerator chip of Claim 59 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted.

65. A method of operating an accelerator chip comprising:

in a hardware translator unit, calculating the address of intermediate language instructions to execute;

obtaining the intermediate language instructions from a memory;

in the hardware translator unit, converting the intermediate language instructions to native instructions;

providing the native instructions to an execution engine; and

in the execution engine, executing the native instructions, wherein for at least one intermediate language instruction a callback to a separate processor chip running a virtual machine is done to handle the intermediate language instruction.

66. The method of Claim 65 wherein the intermediate language instructions are Java bytecodes.

67. An accelerator chip comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions;

an execution engine adapted to execute the native instructions provided by the hardware translator unit;

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory; and

a graphics acceleration engine adapted to be interconnected to a display, the graphics acceleration engine executing intermediate language instructions concerning a display.

68. The accelerator chip of Claim 67 wherein the intermediate language instructions are Java bytecodes.

69. The accelerator chip of Claim 68 wherein Java based libraries are used.

70. The system of Claim 69 wherein the Java based libraries include Java based programs.

71. The system of Claim 70 wherein the Java based programs are modified Java programs.

72. The accelerator chip of Claim 67 in which the display is an LCD display and the graphics acceleration engine implements an LCD display.

73. The accelerator chip of Claim 72 wherein the graphics acceleration engine implements a Java LCD display library function.

74. A system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including at least one indexed instruction to do an indexed load from or store into an array, the instruction concurrently checking a first register storing an array pointer to see whether it is null.

75. The system of Claim 74 wherein the hardware translator unit and the execution engine are positioned on an accelerator chip.

76. The system of Claim 74 wherein the accelerator chip is positioned between a processor chip and a memory.

77. The system of Claim 74 wherein the intermediate language instructions are Java instructions.

78. The system of Claim 74 wherein the hardware translator unit translates some array loading and array storing instructions so as to use at least one index instruction.

79. A system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including at least one indexed instruction to do an indexed load from or store into an array, the execution engine having a zero checking unit adapted to check whether a first register storing an array pointer to see whether it is null, the null checking unit of the execution engine working concurrently with portions of the execution engine doing the indexed load from or store into an array.

80. The system of Claim 79 in which the intermediate language instructions are Java bytecodes.

81. The system of Claim 79 in which the hardware translator unit and execution engine are on an accelerator chip.

82. An system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including at least one bounds checking instruction, the bounds checking instruction ensuring that an index value stored in a first register is less than or equal to an array length value stored in a second register.

83. The system of Claim 82 wherein the intermediate language instructions are Java bytecodes.

84. The system of Claim 82 in which the hardware translator unit and execution engine are part of an accelerator chip.

85. The system of Claim 84 in which the accelerator chip is positioned between a processor chip and a memory.

86. An system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including an instruction that based on values from the last addition or subtraction stores an 1, 0, or -1 in a register.

87. The system of Claim 86 wherein the values include the N, Z and carry bits.

88. The system of Claim 86 wherein the signed instruction checks the Z and the N bits. If the Z bit is high, a 0 is put in the register. If the Z bit is low and N is low, 1 is put in the register. If the Z bit is low and N is high, a -1 is put in the register.

89. The system of Claim 86 in which an unsigned instruction check is done to check the Z and C bits. If the Z bit is high, a 0 is put in the register. If the Z bit is low, and C is high, 1 is put in the register. If the Z bit is low and the C is low, -1 is put in the register.

90. The system of Claim 86 in which both signed and unsigned checks are done.

91. The system of Claim 86 wherein the hardware translator unit and execution engine are on an accelerator chip.

92. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language, the hardware accelerator including a accelerator of a Java processor for the execution of intermediate language instructions.

93. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language, wherein some instructions generate a callback and get executed on the processor chip.

94. The system of Claim 93 wherein a system of registers is used for transferring information between the SoC and accelerator for callbacks.

95. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, wherein the use of the accelerator chip is in a cell phone or mobile handheld device.

96. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory wherein the accelerator is stacked on the SoC in the same package

97. System of Claim 96 wherein the use of the accelerator chip is in a cell phone or mobile handheld device.

98. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory wherein the accelerator is stacked with one or more memory in the same package.

99. System of Claim 98 wherein the use of the accelerator chip is in a cell phone or mobile handheld device.

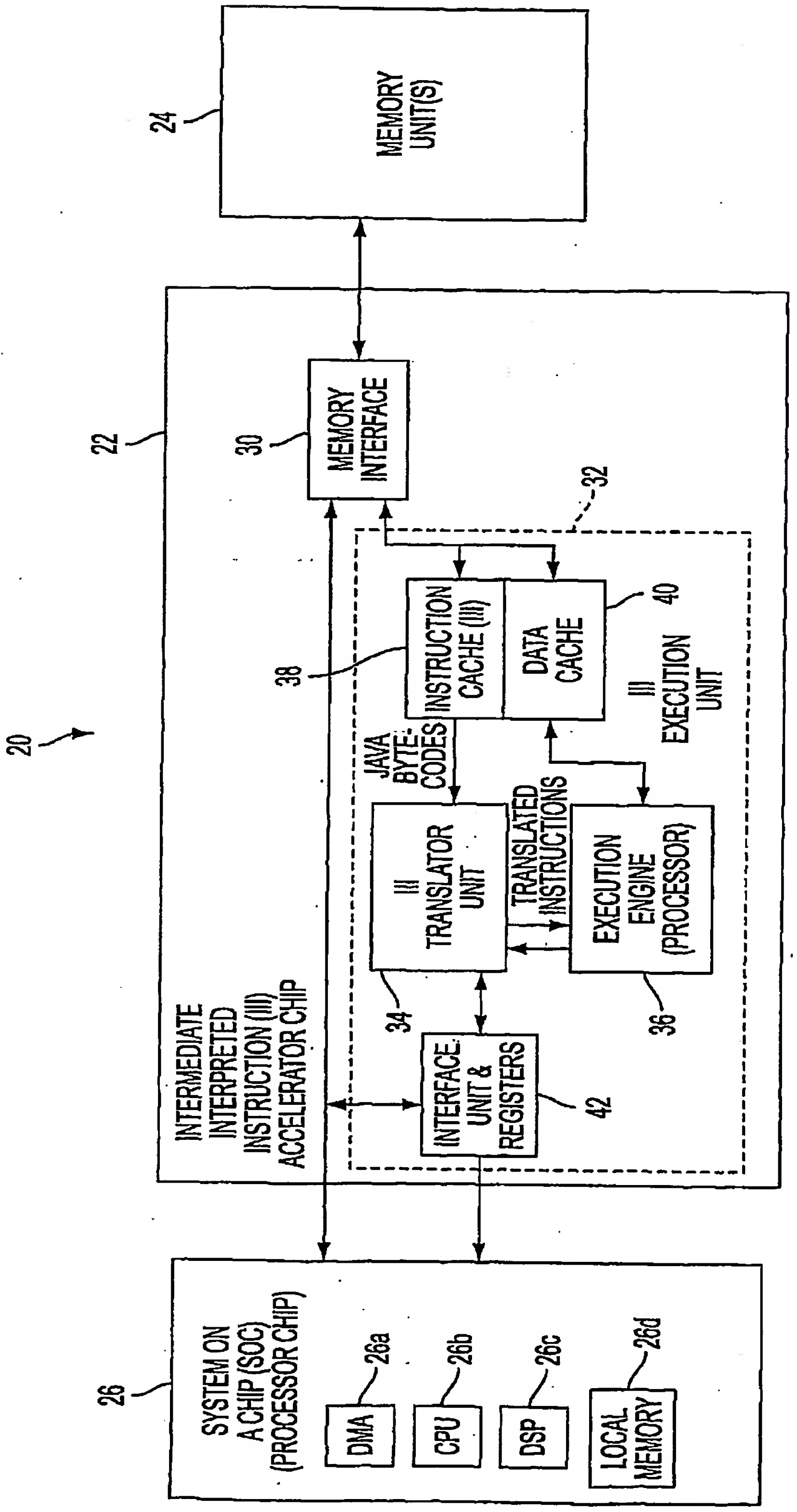
MEMORY UNIT(S)
24
20
22
30
MEMORY INTERFACE
32
INSTRUCTION CACHE (III)
DATA CACHE
40
38
III EXECUTION UNIT
JAVA BYTE-CODES
TRANSLATED INSTRUCTIONS
III TRANSLATOR UNIT
EXECUTION ENGINE (PROCESSOR)
34
36
INTERMEDIATE INTERPRETED INSTRUCTION (III) ACCELERATOR CHIP
INTERFACE UNIT & REGISTERS
42
26
SYSTEM ON A CHIP (SOC) (PROCESSOR CHIP)
DMA
26a
CPU
26b
DSP
26c
LOCAL MEMORY
26d

FIG. 1

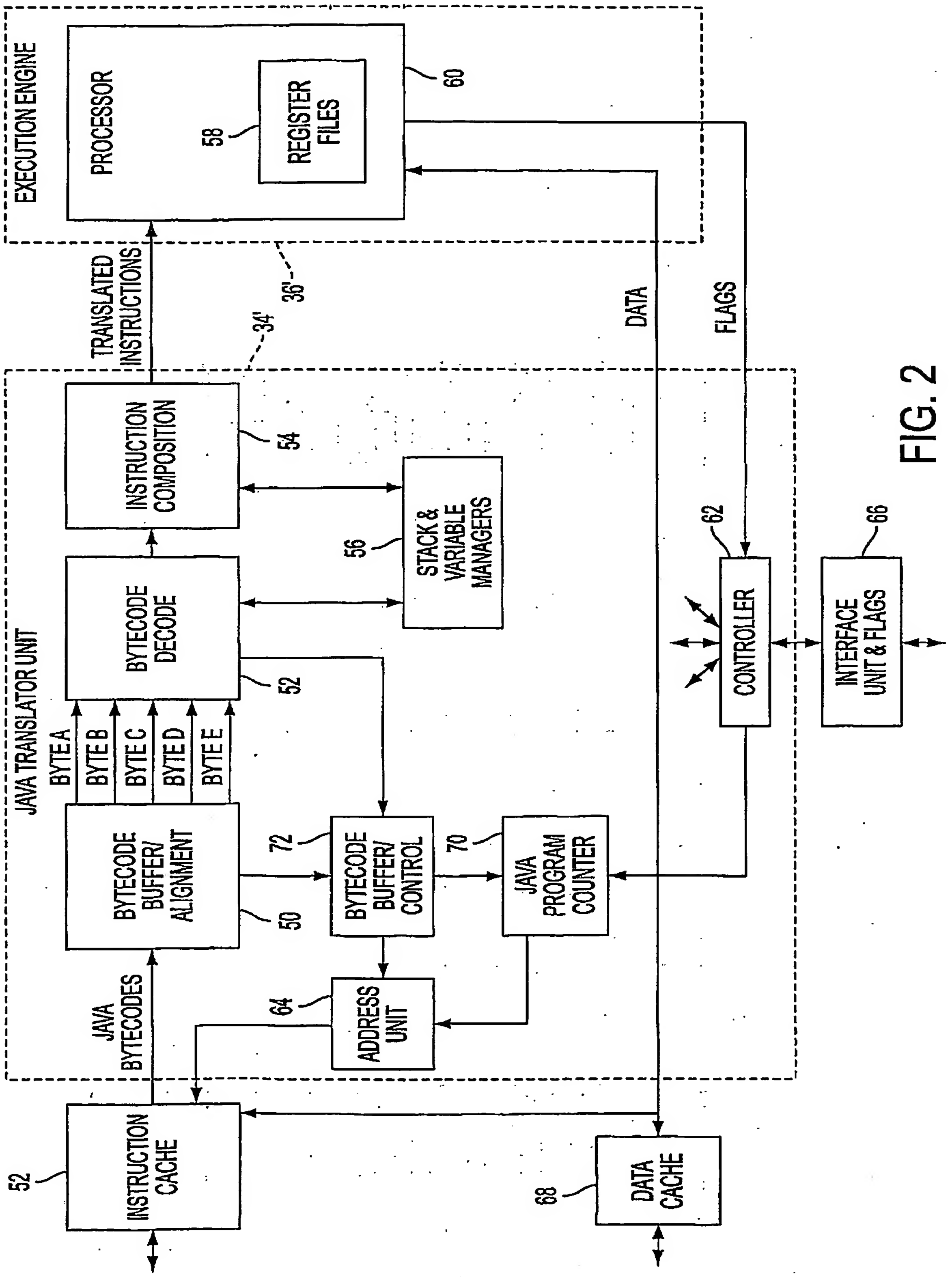
EXECUTION ENGINE
PROCESSOR
REGISTER FILES
58
60
TRANSLATED INSTRUCTIONS
36'
34'
DATA
FLAGS
JAVA TRANSLATOR UNIT
INSTRUCTION COMPOSITION
54
STACK & VARIABLE MANAGERS
56
BYTECODE DECODE
52
BYTE A
BYTE B
BYTE C
BYTE D
BYTE E
BYTECODE BUFFER/ ALIGNMENT
50
BYTECODE BUFFER/ CONTROL
72
JAVA PROGRAM COUNTER
70
ADDRESS UNIT
64
CONTROLLER
62
INTERFACE UNIT & FLAGS
66
JAVA BYTECODES
INSTRUCTION CACHE
52
DATA CACHE
68

FIG. 2

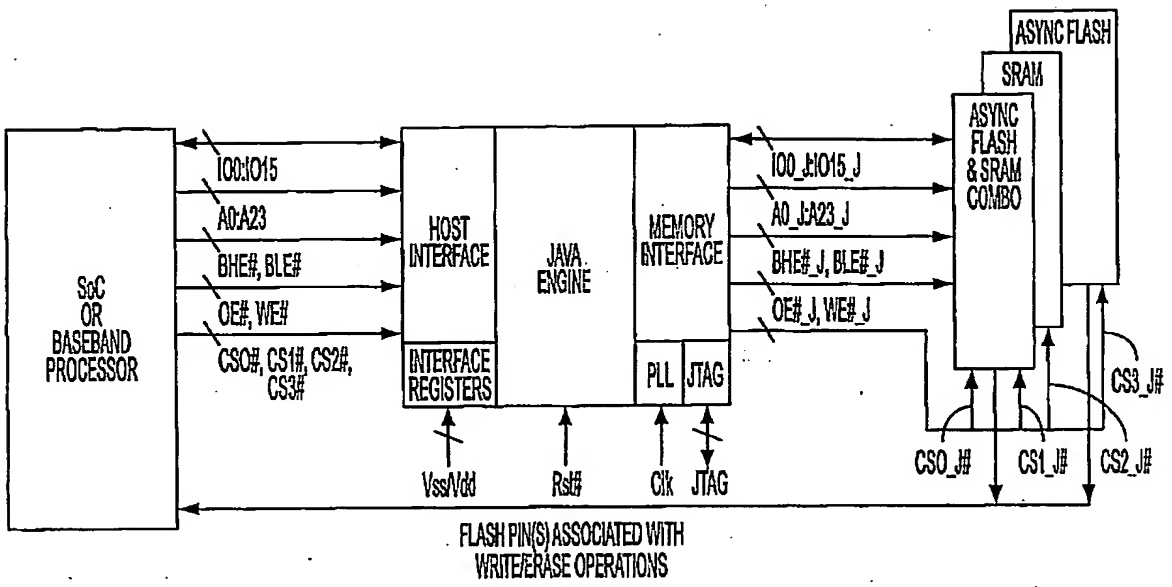
SoC OR BASEBAND PROCESSOR
IO0:IO15
A0:A23
BHE#, BLE#
OE#, WE#
CS0#, CS1#, CS2#, CS3#
HOST INTERFACE
INTERFACE REGISTERS
JAVA ENGINE
MEMORY INTERFACE
PLL
JTAG
Vss/Vdd
Rst#
Clk
JTAG
IO0_J:IO15_J
A0_J:A23_J
BHE#_J, BLE#_J
OE#_J, WE#_J
ASYNC FLASH & SRAM COMBO
SRAM
ASYNC FLASH
CS0_J#
CS1_J#
CS2_J#
CS3_J#
FLASH PIN(S) ASSOCIATED WITH WRITE/ERASE OPERATIONS

FIG. 3

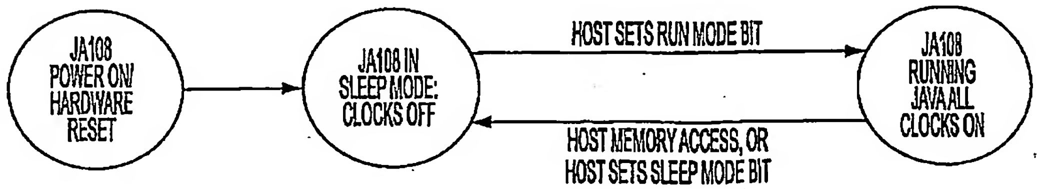
JA108 POWER ON/ HARDWARE RESET
JA108 IN SLEEP MODE: CLOCKS OFF
HOST SETS RUN MODE BIT
JA108 RUNNING JAVA ALL CLOCKS ON
HOST MEMORY ACCESS, OR HOST SETS SLEEP MODE BIT

FIG. 4A

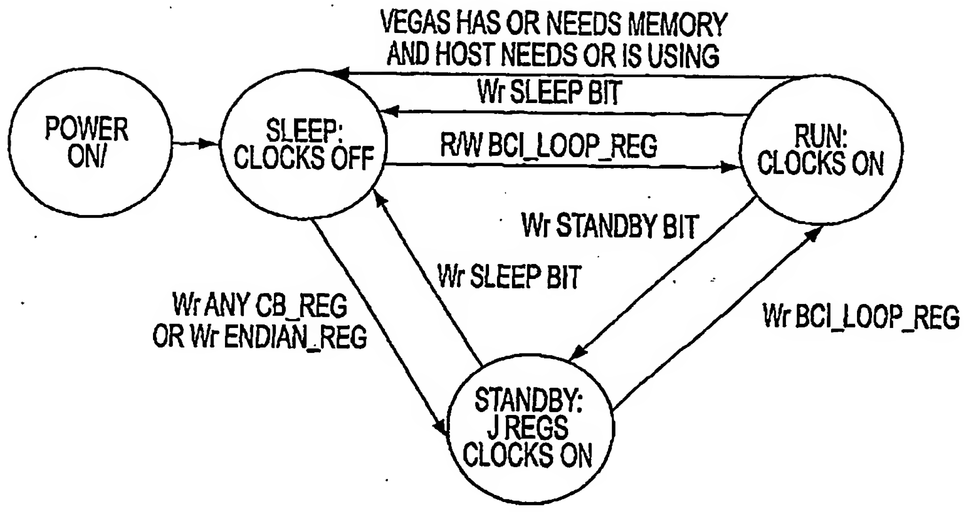
VEGAS HAS OR NEEDS MEMORY AND HOST NEEDS OR IS USING
Wr SLEEP BIT
POWER ON/
SLEEP: CLOCKS OFF
R/W BCI_LOOP_REG
RUN: CLOCKS ON
Wr STANDBY BIT
Wr SLEEP BIT
Wr ANY CB_REG OR Wr ENDIAN_REG
Wr BCI_LOOP_REG
STANDBY: J REGS CLOCKS ON

FIG. 4B

OPERATING STATE	POWER SUPPLY CURRENT	JA108 ENGINE	INTERNAL CACHES	INTERNAL REGISTERS	MEMORY INTERFACE
ON	~0.3mA/MHz	ON	ON	ON	ON
SLEEP	LEAKAGE ONLY	OFF	OFF	OFF	ON

FIG. 5

JAVA BYTECODE INSTRUCTIONS EXECUTED WITHIN THE JA108 CHIP	JAVA BYTECODE INSTRUCTIONS EXECUTED USING SOFTWARE CALLBACKS
CONSTANT LOADS iconst_x, lconst_x, fconst_x, dconst_x, bipush, sipush LOCAL VARIABLE LOAD iload, lload, fload, dload, aload, iload_x, lload_x, fload_x, dload_x, aload_x LOCAL VARIABLE STORE istore, lstore, fstore, dstore, astore, istore_x, lstore_x, fstore_x, dstore_x, astore_x STACK OPERATIONS nop, pop, pop2, dup,dup_x1, dup2, dup_x2, dup2_x1, dup2_x2, swap ARITHMETIC iadd, ladd, isub, lsub, imul, idiv, lrem, ineg, lneg, iinc LOGIC ishl, lshl, ishr, lshr, iushr, lushr, iand, land, ior, lor, ixor, lxor TYPE CONVERSION i2l, l2i, f2d, i2b, i2c, i2s COMPARE lcmp, ifeq, ifne, iflt, ifge, ifgt, ifle, if_icmpeq, if_cmpne, if_cmplt, if_mpge, if_mpgt, if_icmple, if_acmpeq, if_acmpne, ifnull, ifnonull UNCONDITIONAL BRANCH goto, goto_w, jsr, jsrw, ret, tableswitch, lookupswitch ARRAY arraylength, iaload, laload, faload, daload, aaload, baload, caload, saload, iastore, lastore, fastore, dastore, aastore, bastore, castore, sastore QUICKCODES ldc_quick, ldc_w_quick, ldc2_w_quick, getstatic_quick, getstatic2_quick, putstatic_quick, putstatic2_quick, getfield_quick, getfield2_quick, getfield_quick_w, putfield_quick, putfield2_quick, putfield_quick_w	LOAD FROM CONSTANT POOL ldc, ldc_w, ldc2_w * ARITHMETIC FLOATING POING fadd, fsub, fmul, fdiv, frem, fneg * ARITHMETIC DOUBLE dadd, dsub, dmul, ddiv, drem, dneg ARITHMETIC LONGS lmul, ldiv, lrem * TYPE CONVERSION i2f, i2d, l2f, l2d, f2i, f2l, d2i, d2f, d2l * COMPARE fcmpl, fcmpg, dcmpl, dcmpg RETURN ireturn, lreturn, freturn, dreturn, areturn, return STATIC FIELD ACCESS getstatic, putstatic RESERVED BREAKPOINT * INDICATES THAT THE JAVA BYTECODE INSTRUCTION IS NOT PART OF THE CLDC SPECIFICATION

FIG. 6

TYPICAL SYSTEM MEMORY ORGANIZATION

SRAM	FLASH
• JAVA HEAP • JAVA STACK • DOWNLOADED JAVA CLASS FILES (INCLUDING THE CONSTANT POOL) • DOWNLOADED JAVA APPLETS/APPLICATIONS	• JAVA RUNTIME ENVIRONMENT (JRE) - JAVA VIRTUAL MACHINE (JVM, CVM,KVM) - JAVA CLASS LIBRARIES - PROFILE CLASS LIBRARIES • DOWNLOADED JAVA CLASS FILES (INCLUDING THE CONSTANT POOL) INTENDED TO BE SAVED AS RESIDENT • DOWNLOADED JAVA APPLETS/APPLICATIONS INTENDED TO BE SAVED AS RESIDENT • OPERATING SYSTEM & DEVICE DRIVERS • NATIVE RESIDENT APPLICATIONS

FIG. 7

JAVA ACCELERATION PIN FUNCTIONS, ORGANIZED BY FUNCTION

PIN	FUNCTION
IO0:IO15	DATA IN/OUT FROM HOST CPU
A0:A23	ADDRESS INPUTS FROM HOST CPU
BHE#, BLE#	BYTE HIGH ENABLE, BYTE LOW ENABLE FROM HOST (ACTIVE LOW)
OE#, WE#	OUTPUT ENABLE, WRITE ENABLE FROM HOST (ACTIVE LOW)
CS0#, CS1#, CS2#, CS3#	CHIP SELECTS 0, 1, 2, 3 FROM HOST (ACTIVE LOW)
Rst#	RESET (ACTIVE LOW)
Clk	CLOCK INPUT
TDI, TDO, TCK, TMS	JTAG SIGNALS: TEST DATA IN, TEST DATA OUT, TEST CLOCK, TEST MODE SELECT
IO0_J:IO15_J	DATA IN/OUT TO MEMORY FROM THE JA108
A0_J:A18_J	ADDRESS OUT TO MEMORY FROM THE JA108
BHE#_J, BLE#_J	BYTE HIGH AND LOW ENABLE TO MEMORY FROM THE JA108
OE#_J, WE#_J	OUTPUT ENABLE, WRITE ENABLE TO MEMORY FROM THE JA108
CS0#_J, CS1#_J, CS2#_J, CS3#_J	CHIP SELECTS 0, 1, 2, 3 TO MEMORY FROM THE JA108
CB	CALLBACK (ACTIVE HIGH) USED TO SIGNAL THAT A BYTECODE NEEDS TO BE EXECUTED BY SOFTWARE RUNNING ON THE HOST MICROPROCESSOR
Vdd,Vss	POWER AND GROUND

FIG. 8

JA108 ROUND TRIP DELAY (ns): 8, ASSUMED AT .25 MICRON
MEMORY WAIT STATES FOR ACCESS TIMES (IN ns) OF:

BUS SPEED IN MHz	ns	25 PLAIN	25 JA108	45 PLAIN	45 JA108	55 PLAIN	55 JA108	70 PLAIN	70 JA108	85 PLAIN	85 JA108	90 PLAIN	90 JA108	100 PLAIN	100 JA108
20	50.0	0	0	0	1	1	1	1	1	1	1	1	1	1	2
25	40.0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
27	37.0	0	0	1	1	1	1	1	2	2	2	2	2	2	2
30	33.3	0	0	1	1	1	1	2	2	2	2	2	2	2	3
33	30.3	0	1	1	1	1	2	2	2	2	3	2	3	3	3
35	28.6	0	1	1	1	1	2	2	2	2	3	3	3	3	3
37	27.0	0	1	1	1	2	2	2	2	3	3	3	3	3	3
39	25.6	0	1	1	2	2	2	2	3	3	3	3	3	3	4
40	25.0	0	1	1	2	2	2	2	3	3	3	3	3	3	4
43	23.3	1	1	1	2	2	2	3	3	3	3	3	4	4	4
50	20.0	1	1	2	2	2	3	3	3	4	4	4	4	4	5
54	18.5	1	1	2	2	2	3	3	4	4	5	4	5	5	5
56	17.9	1	1	2	2	3	3	3	4	4	5	5	5	5	6
60	16.7	1	1	2	3	3	3	4	4	5	5	5	5	5	6

NOTE: <u>PLAIN</u> IS WITHOUT JA108, AND <u>JA108</u> IS WITH IT (ADD JA108 DELAY). NOTE: JA108'S OWN MEMORY ACCESS ARE AT THE "PLAIN" RATE.

		ACCESS TIME ns (PAGE MODE, LEN)	PKG
INTEL COMBO SIZES: 32Mb/4Mb, 64Mb/8Mb: PN 28F6408W30,28F3204W30	FLASH SRAM	70 (25,4) 70	FBGA-80
TOSHIBA COMBO SIZE 64Mb/8Mb: PN TH50VSF3680/3681AASB	FLASH SRAM	90 90 rd / 70 wr	FBGA-69
AMD COMBO SIZE 32Mb/8Mb: PN Am29DL323D /DS42587	FLASH SRAM	85 85	FBGA-73
SST COMBO SIZES 8Mb/256Kb, 16Mb/256Kb, 16Mb/512Kb: PN SST32HHF802, SST32HHF162, SST32HHF164	FLASH SRAM	70 OR 90 70 OR 90	FBGA-48

FIG. 9

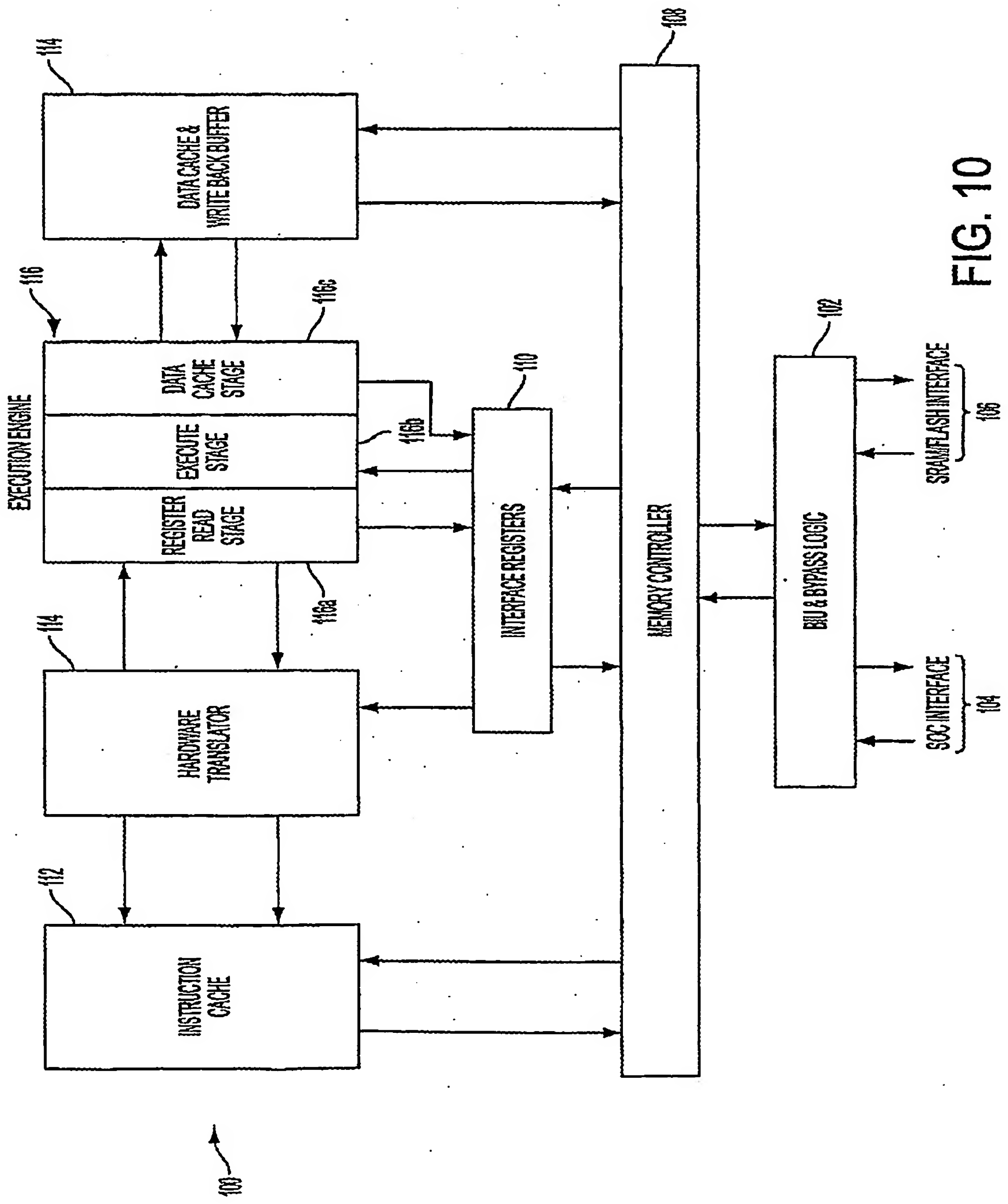
114
DATA CACHE & WRITE BACK BUFFER
116
EXECUTION ENGINE
116c
DATA CACHE STAGE
EXECUTE STAGE
116b
REGISTER READ STAGE
116a
114
HARDWARE TRANSLATOR
112
INSTRUCTION CACHE
110
INTERFACE REGISTERS
108
MEMORY CONTROLLER
102
BIU & BYPASS LOGIC
SRAM/FLASH INTERFACE
106
SOC INTERFACE
104
100

FIG. 10

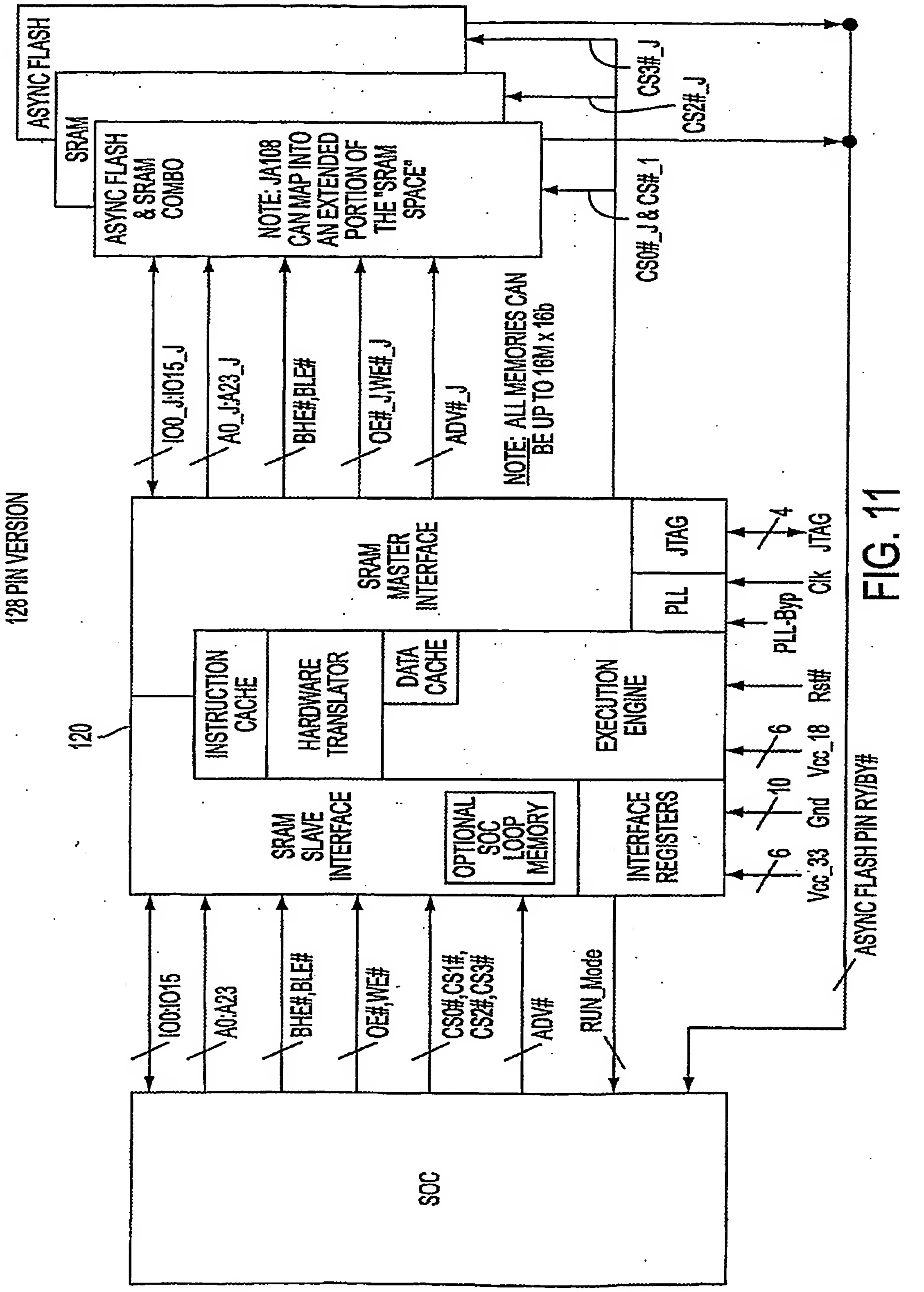
128 PIN VERSION
ASYNC FLASH
SRAM
ASYNC FLASH & SRAM COMBO
NOTE: JA108 CAN MAP INTO AN EXTENDED PORTION OF THE "SRAM SPACE"
CS3#_J
CS2#_J
CS0#_J & CS#_1
IO0_J:IO15_J
A0_J:A23_J
BHE#,BLE#
OE#_J,WE#_J
ADV#_J
NOTE: ALL MEMORIES CAN BE UP TO 16M x 16b
SRAM MASTER INTERFACE
JTAG
PLL
4
JTAG
Clk
PLL-Byp
Rst#
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
120
6
Vcc_18
10
Gnd
6
Vcc_33
SRAM SLAVE INTERFACE
OPTIONAL SOC LOOP MEMORY
INTERFACE REGISTERS
ASYNC FLASH PIN RY/BY#
IO0:IO15
A0:A23
BHE#,BLE#
OE#,WE#
CS0#,CS1#, CS2#,CS3#
ADV#
RUN_Mode
SOC

FIG. 11

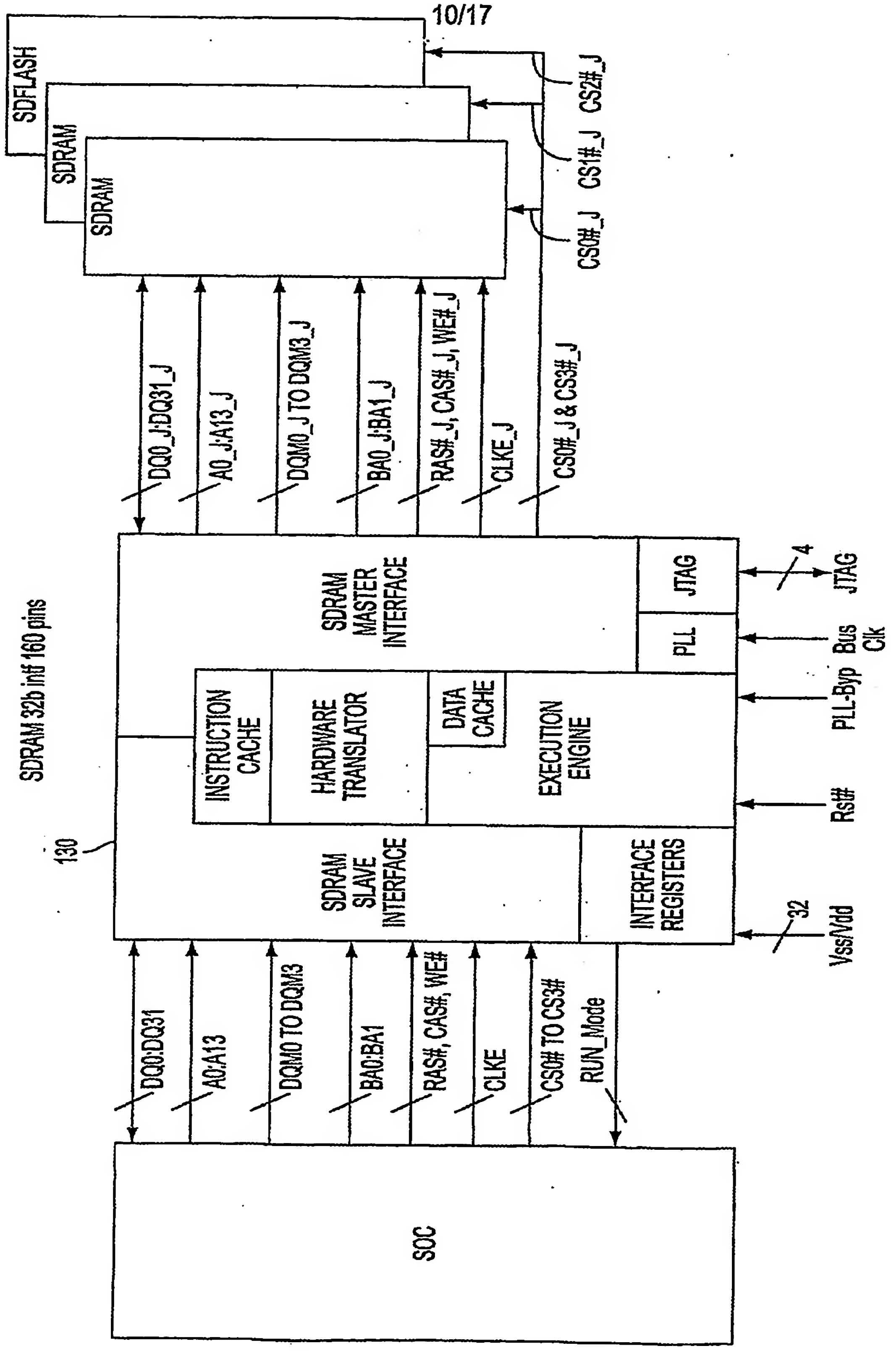
SDFLASH
SDRAM
SDRAM
CS0#_J
CS1#_J
CS2#_J
DQ0_J:DQ31_J
A0_J:A13_J
DQM0_J TO DQM3_J
BA0_J:BA1_J
RAS#_J, CAS#_J, WE#_J
CLKE_J
CS0#_J & CS3#_J
SDRAM 32b intf 160 pins
130
SDRAM MASTER INTERFACE
JTAG
PLL
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
SDRAM SLAVE INTERFACE
INTERFACE REGISTERS
4
JTAG
Bus Clk
PLL-Byp
Rst#
32
Vss/Vdd
DQ0:DQ31
A0:A13
DQM0 TO DQM3
BA0:BA1
RAS#, CAS#, WE#
CLKE
CS0# TO CS3#
RUN_Mode
SOC

FIG. 12

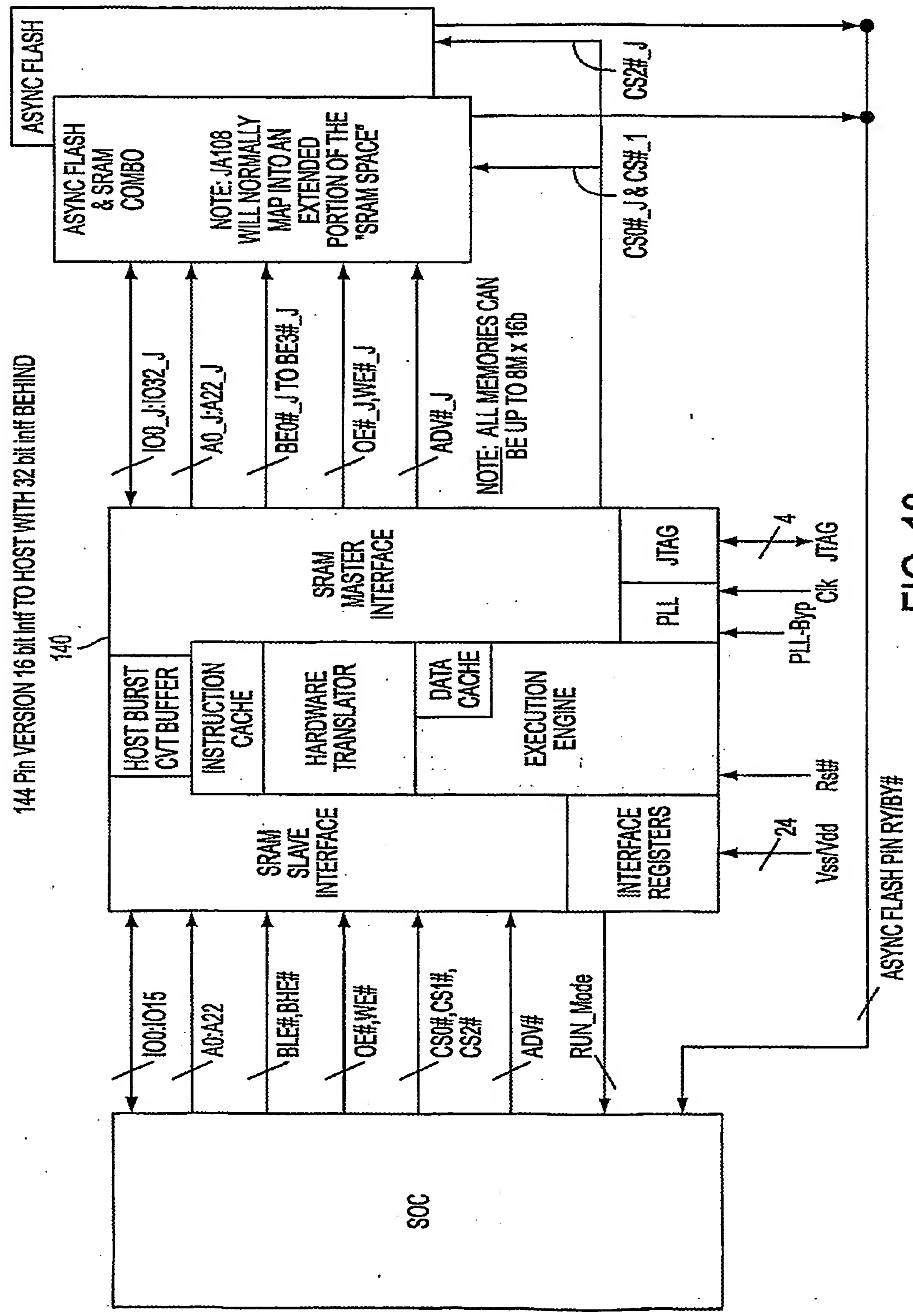
144 Pin VERSION 16 bit Intf TO HOST WITH 32 bit intf BEHIND
140
ASYNC FLASH
ASYNC FLASH & SRAM COMBO
NOTE: JA108 WILL NORMALLY MAP INTO AN EXTENDED PORTION OF THE "SRAM SPACE"
CS2#_J
CS0#_J & CS#_1
IO0_J:IO32_J
A0_J:A22_J
BE0#_J TO BE3#_J
OE#_J,WE#_J
ADV#_J
NOTE: ALL MEMORIES CAN BE UP TO 8M x 16b
SRAM MASTER INTERFACE
JTAG
PLL
4
JTAG
Clk
PLL-Byp
HOST BURST CVT BUFFER
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
Rst#
SRAM SLAVE INTERFACE
INTERFACE REGISTERS
24
Vss/Vdd
ASYNC FLASH PIN RY/BY#
IO0:IO15
A0:A22
BLE#,BHE#
OE#,WE#
CS0#,CS1#, CS2#
ADV#
RUN_Mode
SOC

FIG. 13

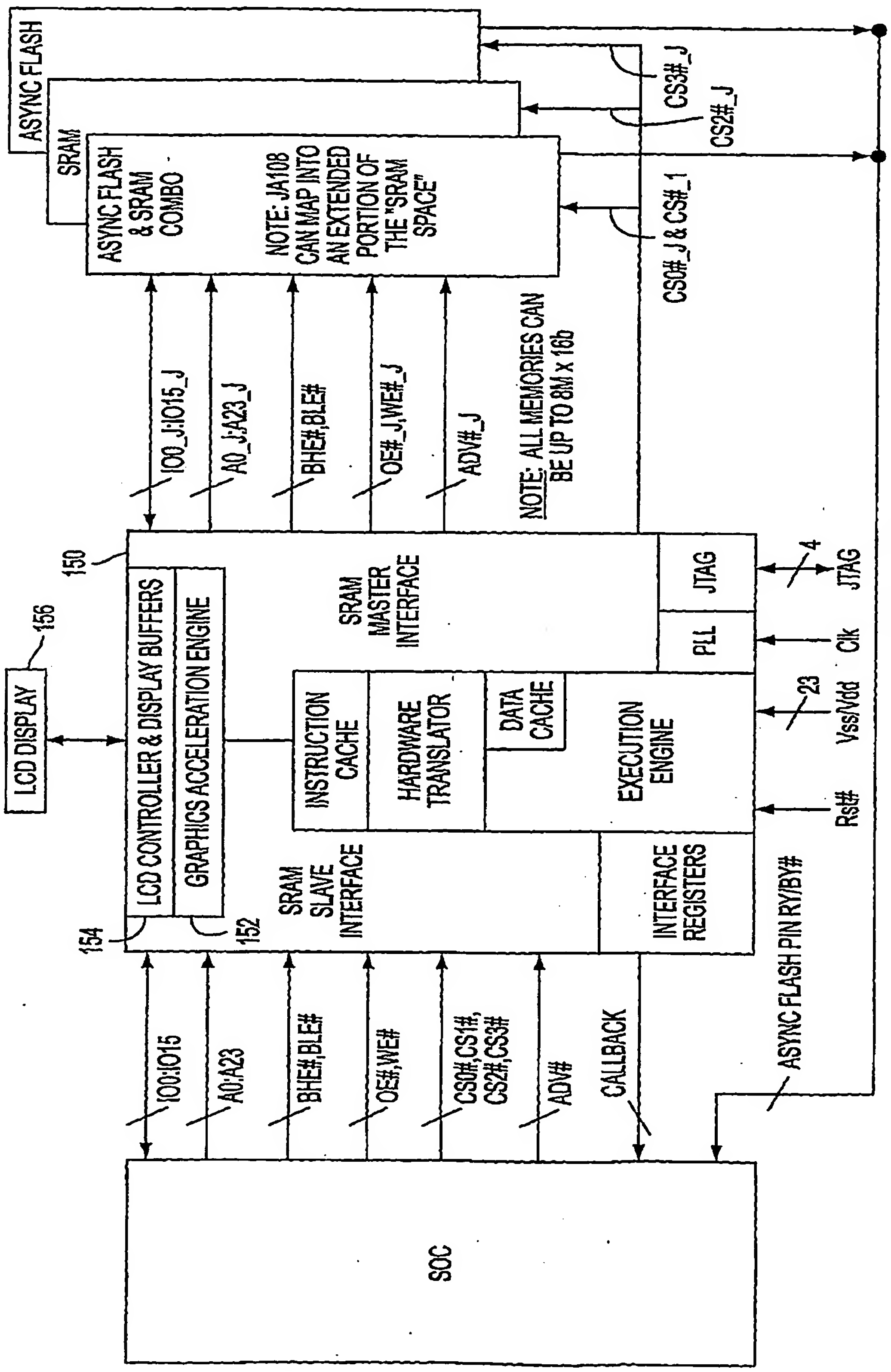
ASYNC FLASH
SRAM
ASYNC FLASH & SRAM COMBO
NOTE: JA108 CAN MAP INTO AN EXTENDED PORTION OF THE "SRAM SPACE"
CS3#_J
CS2#_J
CS0#_J & CS#_1
IO0_J:IO15_J
A0_J:A23_J
BHE#,BLE#
OE#_J,WE#_J
ADV#_J
NOTE: ALL MEMORIES CAN BE UP TO 8M x 16b
150
156
LCD DISPLAY
LCD CONTROLLER & DISPLAY BUFFERS
GRAPHICS ACCELERATION ENGINE
SRAM MASTER INTERFACE
JTAG
PLL
4
JTAG
Clk
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
23
Vss/Vdd
Rst#
SRAM SLAVE INTERFACE
INTERFACE REGISTERS
154
152
IO0:IO15
A0:A23
BHE#,BLE#
OE#,WE#
CS0#,CS1#, CS2#,CS3#
ADV#
CALLBACK
ASYNC FLASH PIN RY/BY#
SOC

FIG. 14

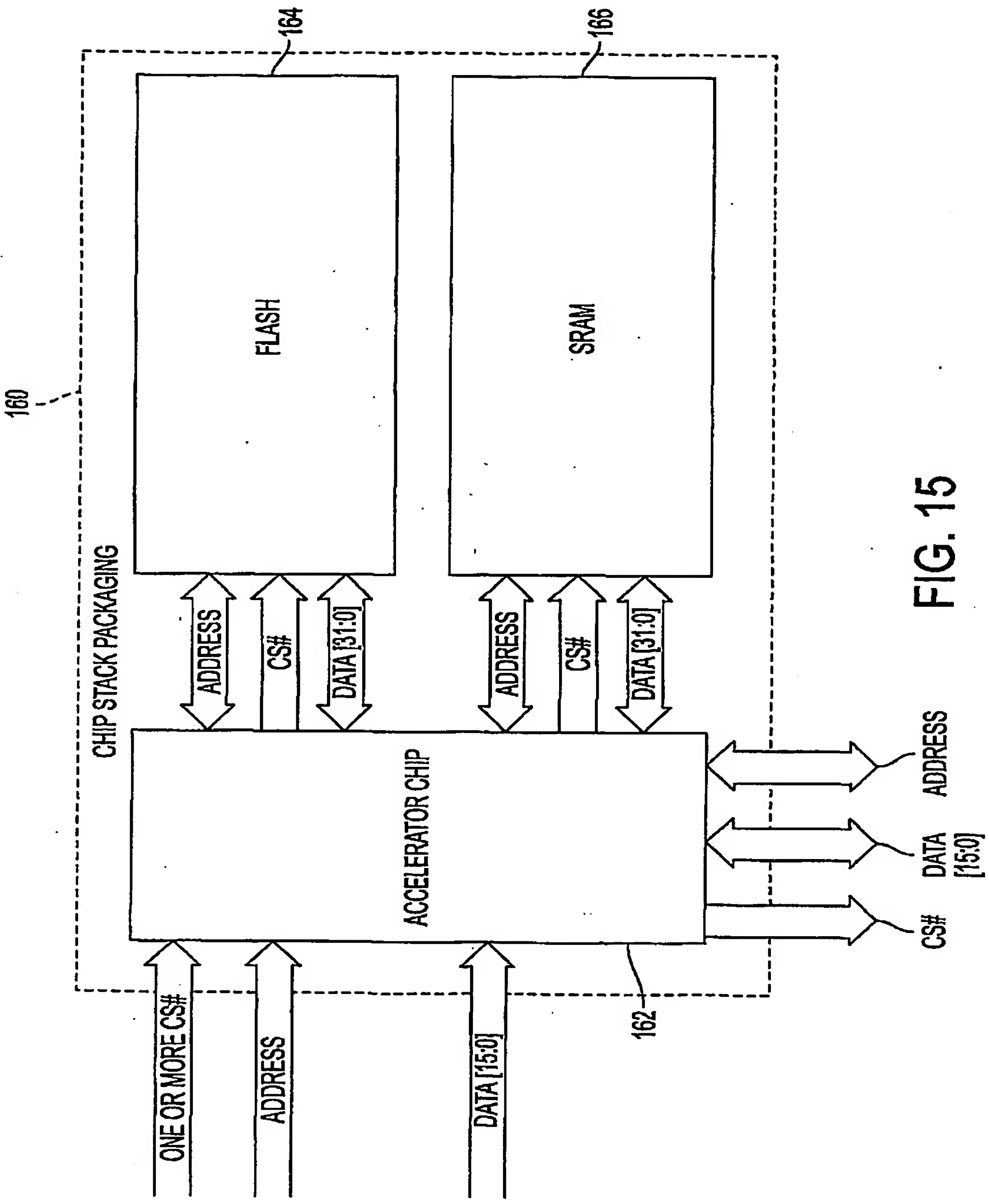
164
166
160
CHIP STACK PACKAGING
FLASH
SRAM
ADDRESS
CS#
DATA [31:0]
ADDRESS
CS#
DATA [31:0]
ACCELERATOR CHIP
ADDRESS
DATA [15:0]
CS#
162
ONE OR MORE CS#
ADDRESS
DATA [15:0]

FIG. 15

SGTLTO	BASED ON C/Z FROM LAST SIGNED ADD/SUB WRITE -1, 0, 1 IN Rd	
SGTLTOU	BASED ON N/Z FROM LAST UNSIGNED ADD/SUB WRITE -1, 0, 1 IN Rd	
LDxNC	LOAD UNSIGNED WORD REGISTER INDEXED	(CHECK REG FOR NULL)
LWxNC	LOAD WORD IMMEDIATE INDEXED	(CHECK REG FOR NULL)
STxNC	STORE WORD REGISTER INDEXED	(CHECK REG FOR NULL)
SWxNC	STORE WORD INDEXED	(CHECK REG FOR NULL)
BNDCK	DO BOUNDS CHECK	

FIG. 16A

SGTLTO

FROM LAST SUB/ADD

N	Z	OUTPUT
x	1	0
0	0	1
1	0	-1

FIG. 16B

SGTLTOU

FROM LAST SUB/ADD

CARRY	Z	OUTPUT
x	1	0
0	0	-1
1	0	1

FIG. 16C

BNDCK

SUBTRACT INDEX FROM ARRAY SIZE

CARRY	
0	NO EXCEPTION
1	EXCEPTION

FIG. 16D

LDxNC
LWxNC
STxNC
SWxNC

ARRAY POINTER	
0	EXCEPTION
NOT ZERO	NO EXCEPTION

FIG. 16E

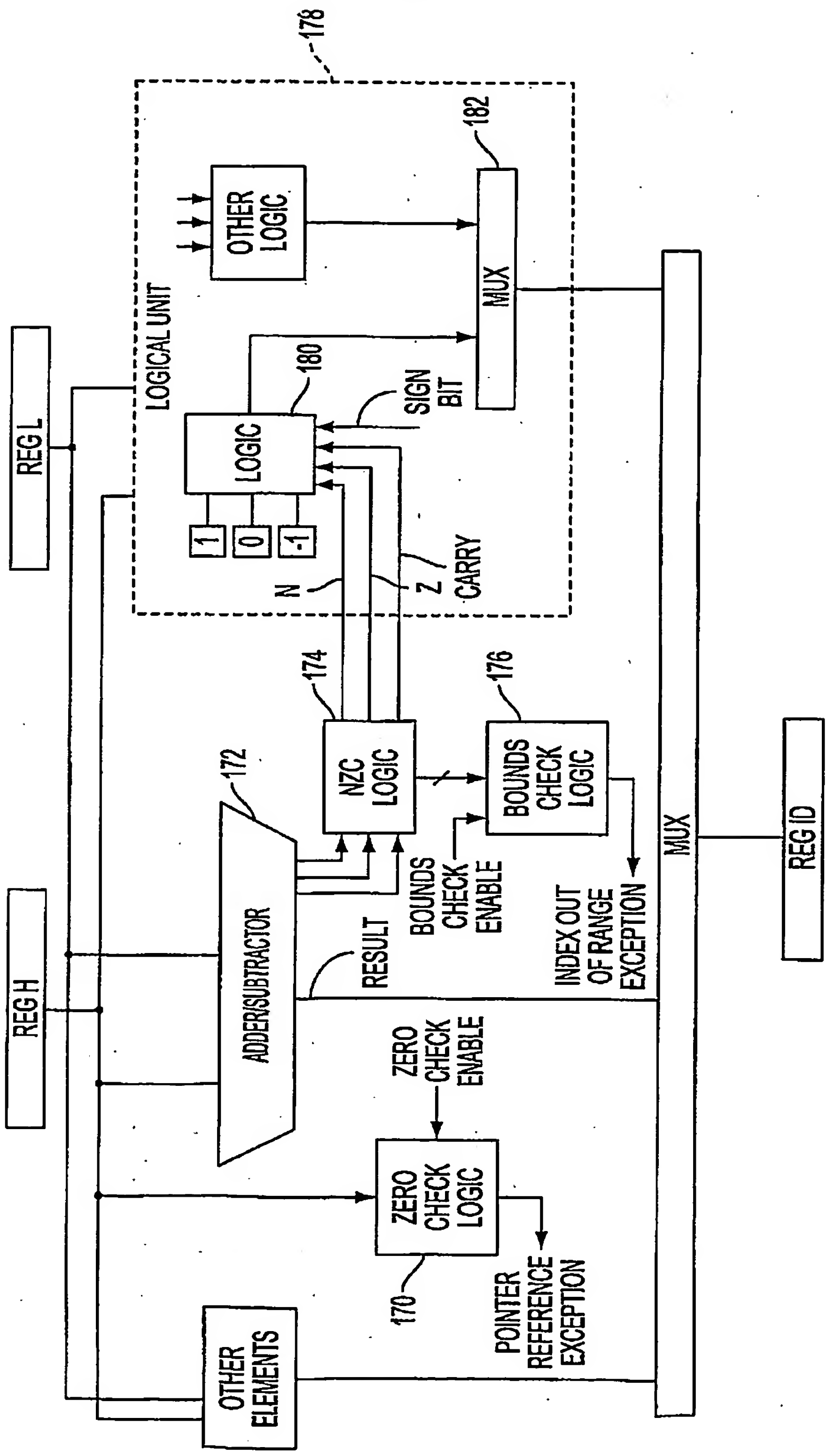
178
LOGICAL UNIT
OTHER LOGIC
182
MUX
180
LOGIC
SIGN BIT
1
0
-1
N
Z
CARRY
REG L
REG H
174
NZC LOGIC
176
BOUNDS CHECK LOGIC
172
ADDER/SUBTRACTOR
RESULT
BOUNDS CHECK ENABLE
INDEX OUT OF RANGE EXCEPTION
MUX
REG ID
ZERO CHECK ENABLE
ZERO CHECK LOGIC
170
POINTER REFERENCE EXCEPTION
OTHER ELEMENTS

FIG. 17

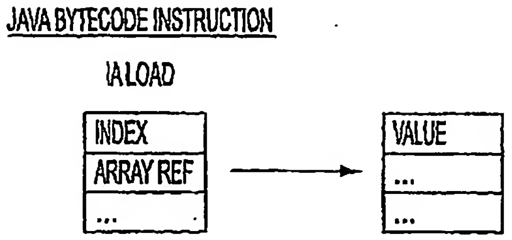

FIG. 18A

CONVENTIONAL MICROCODE

```
BEQ     rSRC2, r00, RAISE NULL POINTER EXCEPTION
LWX     rTEMP1, ARRAY_LENGTH_INDEX (rSRC2)
BLT     rSRC1, r00, RAISE A00B EXCEPTION
BGT     rSRC1, rTEMP1, RAISE A00B EXCEPTION
ADDIV   rTEMP1, rSRC2, ARRAY_BASE_BYTE_OFFSET
LDWX    rRes, rSRCI (rTEMP1)
```

FIG. 18B

MICROCODE WITH NEW INSTRUCTIONS

```
LWXNC   rTEMP1, ARRAY_LENGTH_INDEX (rSRC2)
BNDCK   rTEMP1, rSRC1
ADDIV   rTEMP1, rSRC2, ARRAY_BASE_BYTE_OFFSET
LDWX    rRES, rSRC1 (rTEMP1)
```

FIG. 18C

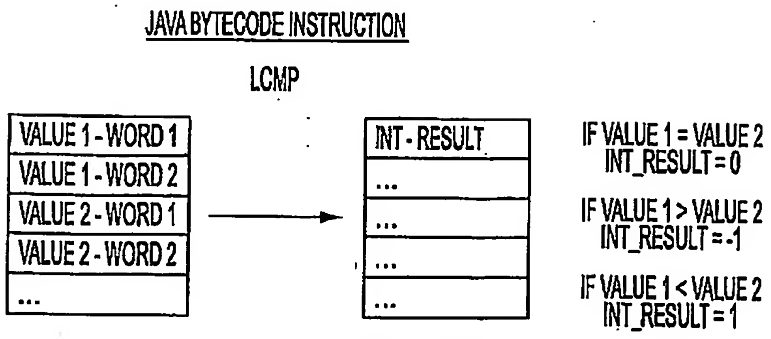

FIG. 19A

CONVENTIONAL MICROCODE

```
                      BEQ   rSRC1W0, rSRC2W0, EQUALMSW
                      BLT   rSRC2W0, rSRC1W0, NEGATIVE RESULT
    POSITIVE RESULT:  ORI   rRES, r00, 1
                                  {QUIT}
          EQUAL MSW:  BGE   rSRC2W1, rSRC1W1, GREATER THAN EQUAL
    NEGATIVE RESULT:  ORI   rRES, r00, -1
                                  {QUIT}
GREATER THAN EQUAL:   BGT   rSRC2W1, rSRC1W1, POSITIVE RESULT
                      OR    rRES, r00, r00
```

FIG. 19B

MICROCODE WITH NEW INSTRUCTIONS

```
      SUBU          rTEMP1, rSRC2W0, rSRC1W0
      SGTLTOU       rRES
      BNE           rRES, r00, END
      SUBU          rTEMP1, rSRC2W1, rSRC1W1
      SGTLTOU       rRES
END:          {QUIT}
```

FIG. 19C

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/20992

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06 F 9/44
US CL : 717/118
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 717/118, 717/148, 717/108

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
ACM

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US 6,026,485 A (O'CONNOR et al) 15 February 2002, col. 2-5, lines 60-67 and 1-67	1-99
Y,E	US 6,446,192 B1(NARASIMHAM et al.) 03 September 2002, col 2-4, lines 1-67	1-99

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	
"A" document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E" earlier document published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	"&" document member of the same patent family

Date of the actual completion of the international search 03 SEPTEMBER 2002	Date of mailing of the international search report 18 SEP 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer ANIL KHATRI Peggy Harrod Telephone No. (703) 305-0282